TÉCNICO+
FORMAÇÃO AVANÇADA

Deep Learning

Gonçalo M. Correia

GALP 2023

Optimization for training neural networks

Surrogate losses

- In the previous lecture, we've seen that—ideally—we would like to minimize the risk

$$\mathbb{E}_{(x,y)\sim\mu}\left[L(x, y; W_{1:L})\right]$$

- Unfortunately, since the distribution μ is unknown, we must instead use surrogate loss functions

Surrogate losses

- Examples of surrogate loss functions:
  - Empirical risk,

$$\frac{1}{N}\sum_{n=1}^{N} L(x_n, y_n; W_{1:L})$$

  - Regularized empirical risk,

$$\mathcal{L}(W) = \frac{1}{N}\sum_{n=1}^{N} L(x_n, y_n; \{W_{1:L}\}) + \lambda R(\{W_{1:L}\})$$

Surrogate losses

- Sometimes, when *L* is hard to compute, or not differentiable, we may even replace *L* for a “friendlier” alternative
  - E.g., the 0-1 loss,

$$L_{0-1}(x_n, y_n; W) = \mathbb{I}_W[y_n \neq y(x_n)]$$

is not differentiable, and is often replaced by the negative log likelihood loss,

$$L_{\text{NLL}}(x_n, y_n; W) = -\log \mathbb{P}_W[\text{class} = y_n \mid x_n]$$

Gradient descent

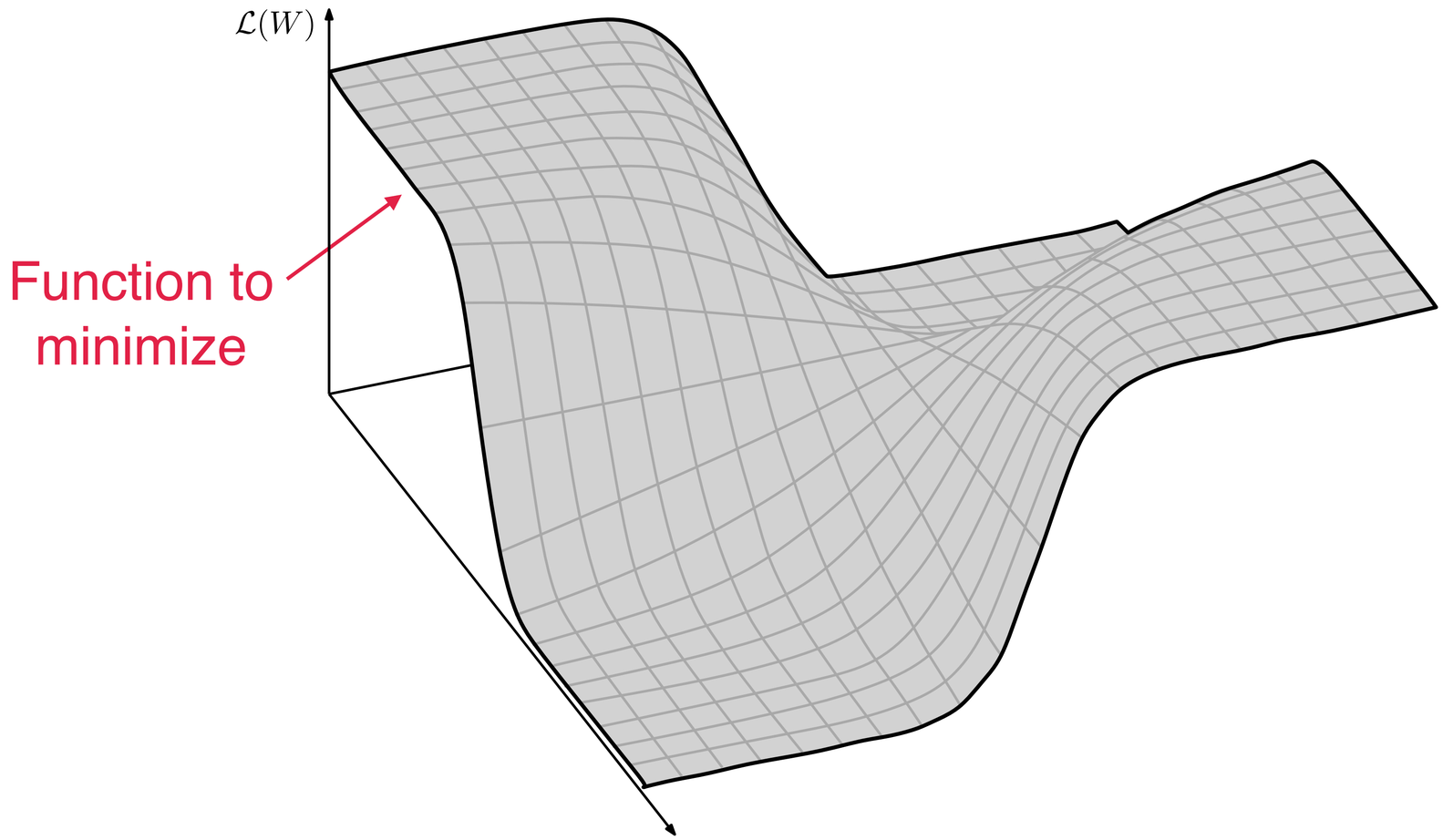
- Let us suppose, for example, that we wish to minimize the regularized empirical risk,

$$\mathcal{L}(W) = \frac{1}{N} \sum_{n=1}^{N} \underbrace{L(x_n, y_n; W_{1:L}) + \lambda R(W_{1:L})}_{\mathcal{L}_n(W)}$$

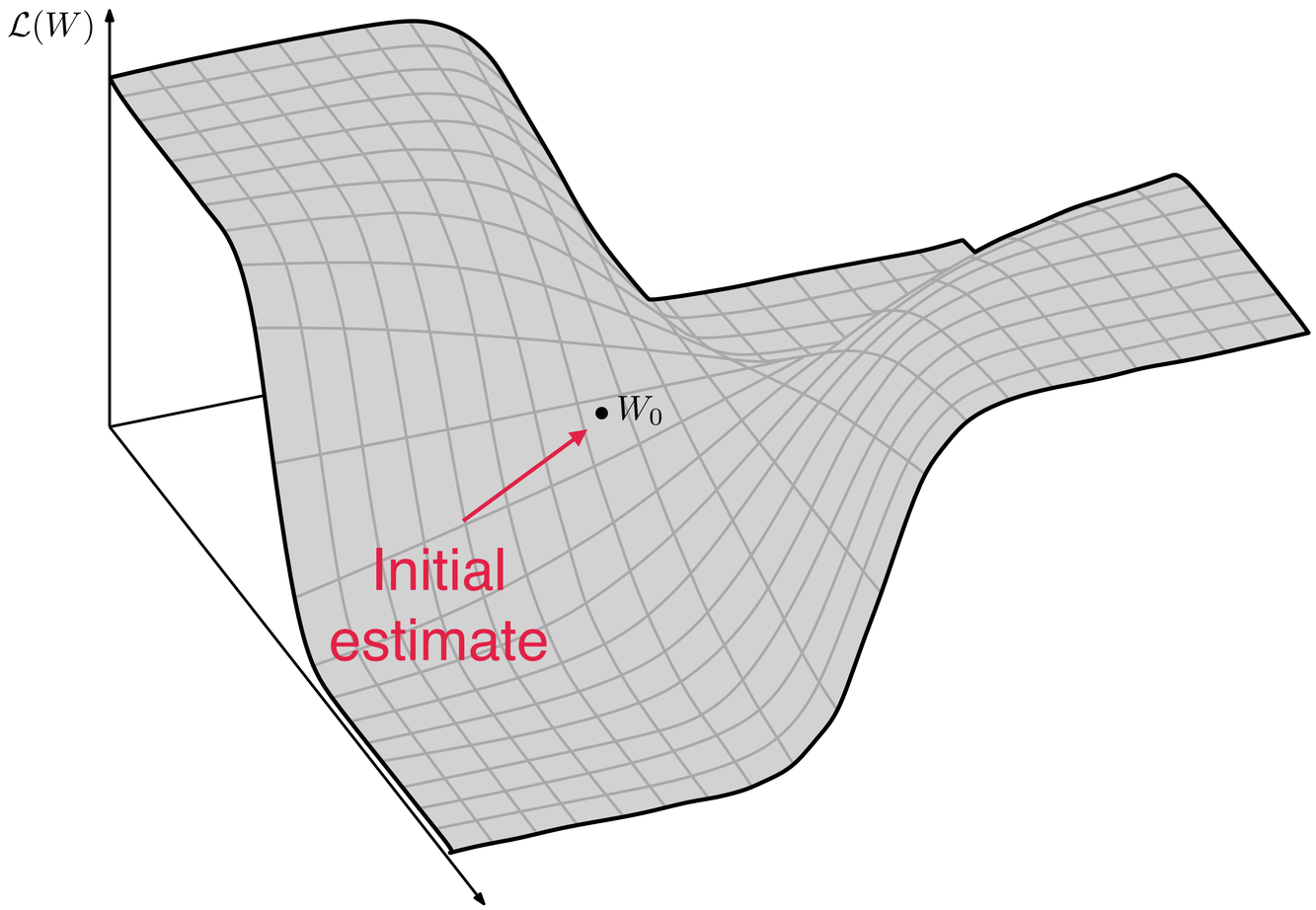
- W is optimized using, for example, gradient descent,

$$W_\ell^{(k+1)} \leftarrow W_\ell^{(k)} - \alpha \sum_{n=1}^{N} \nabla_{W_\ell} \mathcal{L}_n(W), \qquad \ell = 1, \ldots, L$$

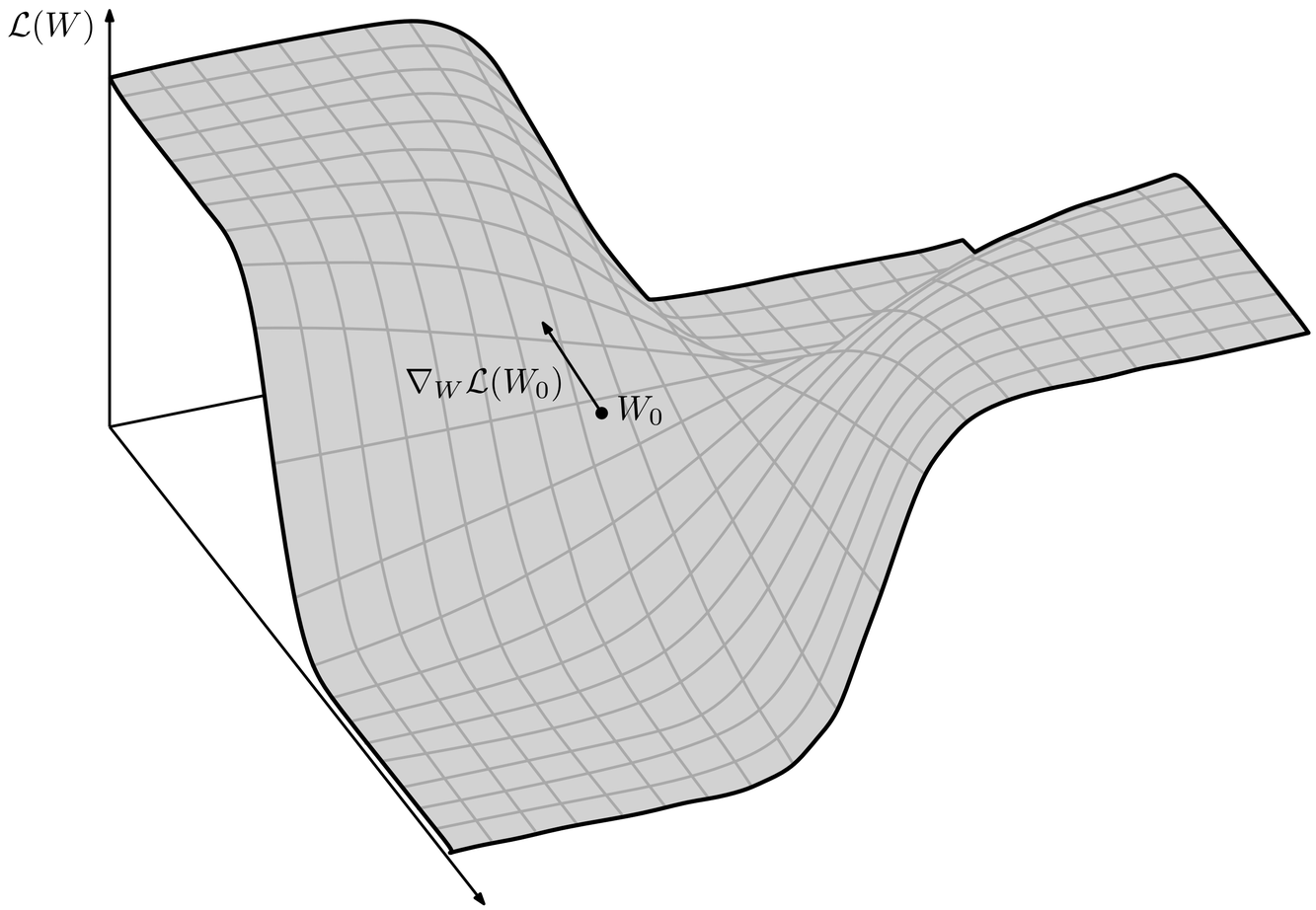
Gradient descent



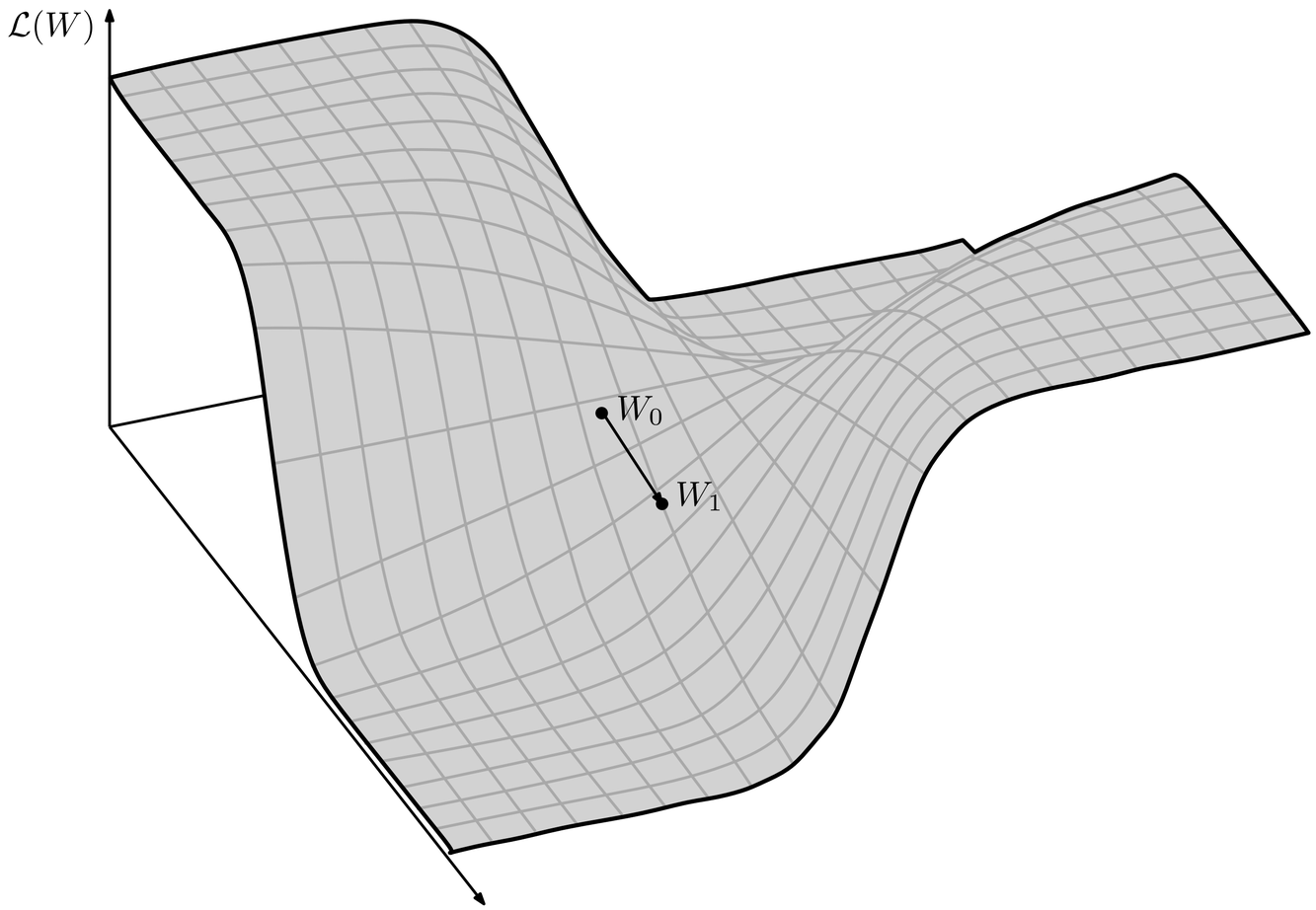
Gradient descent



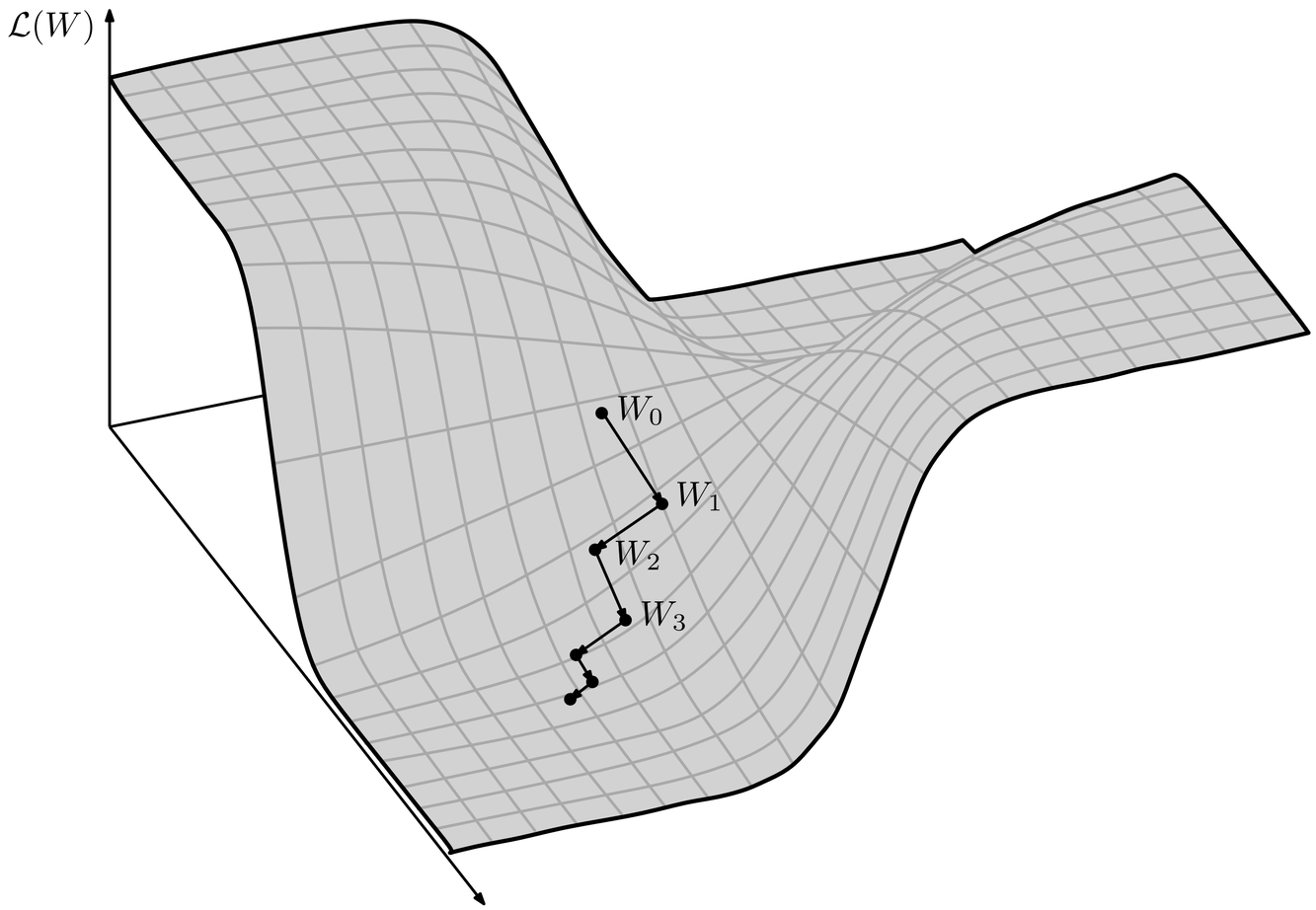
Gradient descent



Gradient descent



Gradient descent



Gradient descent

- However, for large datasets, the gradient

$$\sum_{n=1}^{N} \nabla_{W_\ell} \mathcal{L}_n(W)$$

  may take too long to compute

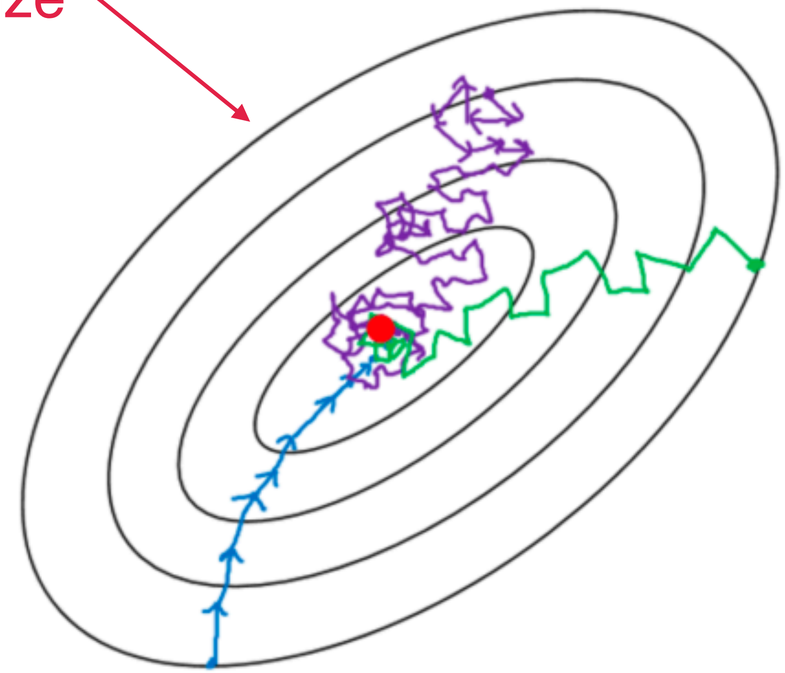
- Instead, we can use mini-batches of samples or even a single sample:

$$W_\ell^{(k+1)} \leftarrow W_\ell^{(k)} - \alpha \nabla_{W_\ell} \mathcal{L}_n(W), \qquad \ell = 1, \ldots, L$$

Stochastic gradient descent

Gradient descent

Level lines of function to minimize



Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

Stochastic gradient descent

- Larger batches provide more accurate gradient estimates, but take longer to compute
- Multi-core architectures can take advantage of the use of mini-batches
  - Each sample can be processed in parallel
  - Too small batches under-exploit parallelization
- Second order methods (that use the Hessian) are more sensitive to the noise introduced by the use of mini-batches

Challenges in NN optimization

What can go wrong?

- Ill-conditioning
- Local minima
- Saddle points, plateaux and flat regions
- Vanishing and exploding gradients
- Inexact gradients

Ill-conditioning

- Why does gradient descent work?
  - We can approximately write

$$\mathcal{L}(W) \approx \mathcal{L}(W_{\text{old}}) + (W - W_{\text{old}})^T \nabla_W \mathcal{L}(W_{\text{old}}) + \text{small stuff}$$

  - Using gradient descent, if we set

$$W = W_{\text{old}} - \alpha \nabla_W \mathcal{L}(W_{\text{old}})$$

Ill-conditioning

- Why does gradient descent work?
  - We can approximately write

$$\mathcal{L}(W) \approx \mathcal{L}(W_{\text{old}}) + (W - W_{\text{old}})^T \nabla_W \mathcal{L}(W_{\text{old}}) + \text{small stuff}$$

  - Using gradient descent, if we set

Replacing here

$$W - W_{\text{old}} = -\alpha \nabla_W \mathcal{L}(W_{\text{old}})$$

Ill-conditioning

- Why does gradient descent work?
  - We can approximately write

$$\mathcal{L}(W) \approx \underbrace{\mathcal{L}(W_{\text{old}}) - \alpha \|\nabla_W \mathcal{L}(W_{\text{old}})\|_2^2}_{<\mathcal{L}(W_{\text{old}})} + \text{small stuff}$$

Ill-conditioning

- But what if the “small stuff” is not so small?
- In other words,

$$\mathcal{L}(W) \approx \mathcal{L}(W_{\text{old}}) - \alpha \|\nabla_W \mathcal{L}(W_{\text{old}})\|_2^2 + \underbrace{\text{small stuff}}_{\text{not so small?...}}$$

Ill-conditioning

- Let's write

$$g = \nabla_W \mathcal{L}(W_{\text{old}})$$

For "gradient"

- Then,

$$\mathcal{L}(W) \approx \mathcal{L}(W_{\text{old}}) - \alpha \|\nabla_W \mathcal{L}(W_{\text{old}})\|_2^2 + \text{small stuff}$$

Ill-conditioning

- Let's write

$$g = \nabla_W \mathcal{L}(W_{\text{old}})$$

- Then,

$$\mathcal{L}(W) \approx \mathcal{L}(W_{\text{old}}) - \alpha g^T g + \text{small stuff}$$

Transpose

Ill-conditioning

- Let's write

$$g = \nabla_W \mathcal{L}(W_{\text{old}})$$

- Then,

$$\mathcal{L}(W) \approx \mathcal{L}(W_{\text{old}}) - \alpha g^T g + \frac{1}{2}\alpha^2 g^T H g + \text{even smaller stuff}$$

Hessian matrix

Ill-conditioning happens if this term is too large

TÉCNICO+
FORMAÇÃO AVANÇADA

Ill-conditioning

- Ill conditioning may occur if the second-order term is too large
  - Technically, if the matrix $\alpha H - I$ is positive definite
- We can mitigate it by decreasing the step size, for example, by setting

$$\alpha < \frac{g^T g}{g^T H g}$$

- This, however, renders learning slow

Finding minima

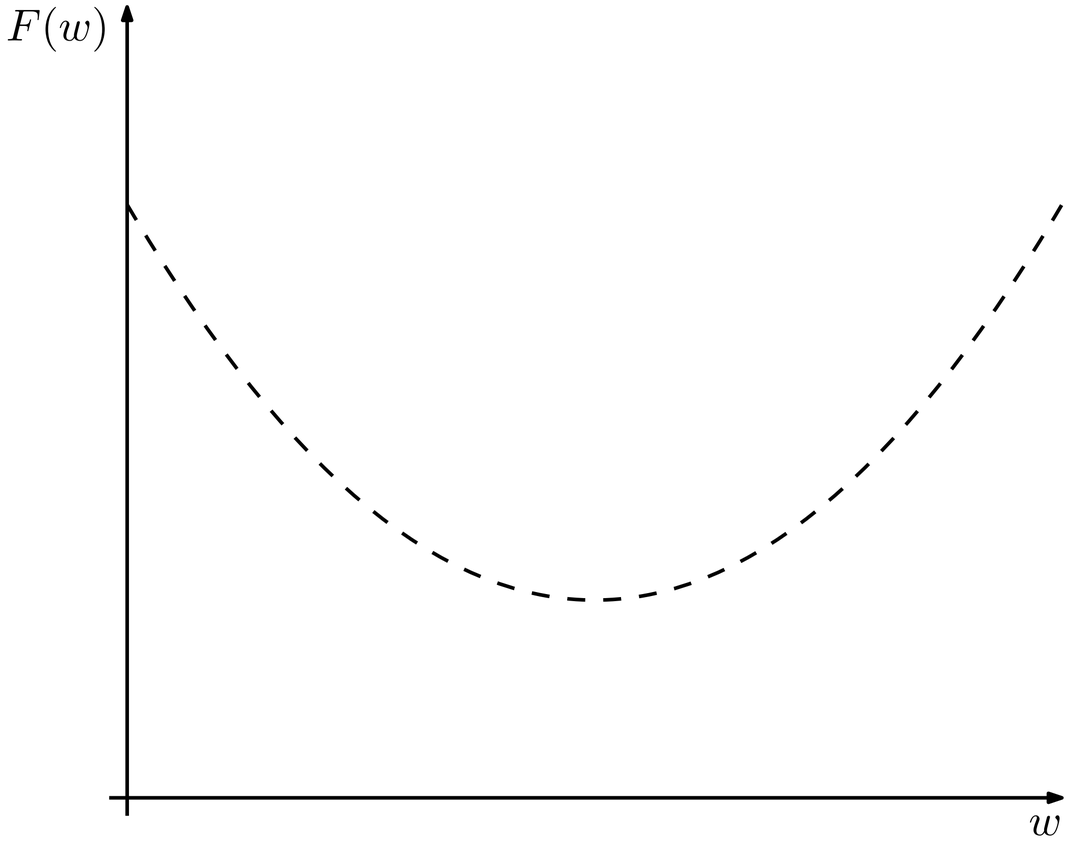
- Gradient descent works by moving down the “slope” of the loss function
- It stops when the gradient is zero
  - Hopefully, it is a minimum point!

Finding minima

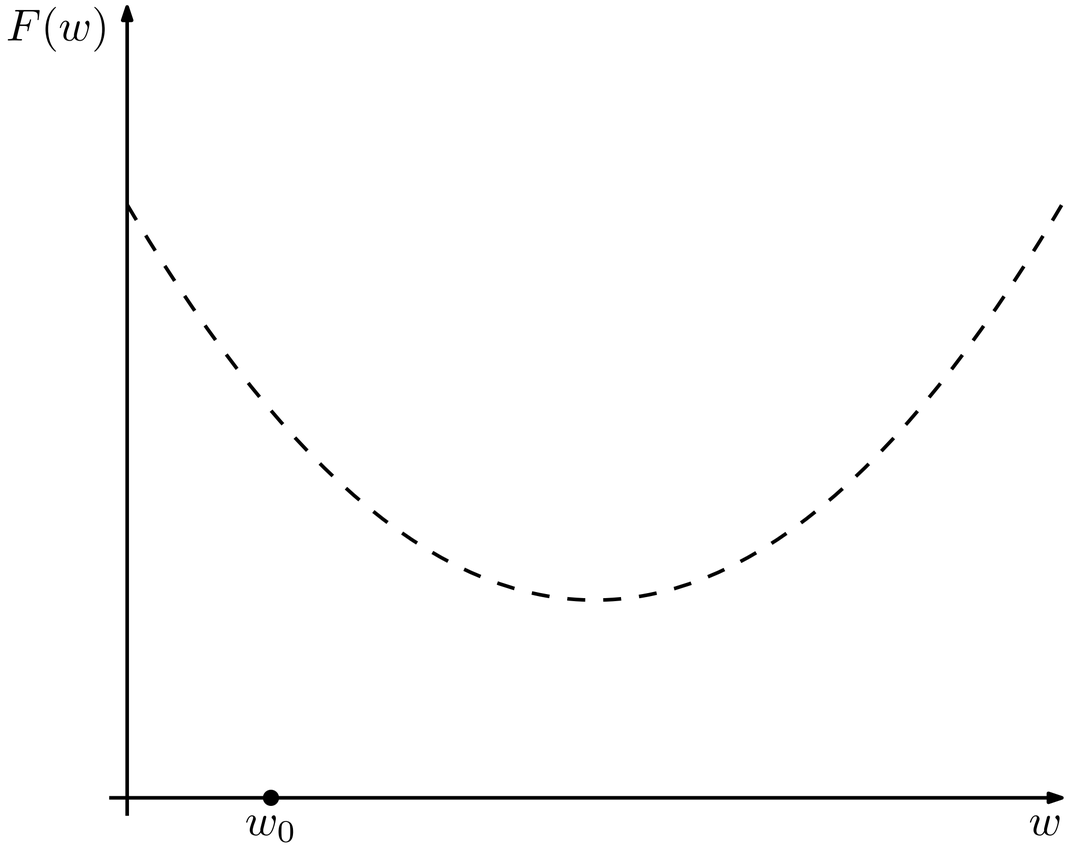
- If the function is convex, perfect!
  - A function *F* is convex if

$$F(\lambda w_0 + (1-\lambda)w_1) \leq \lambda F(w_0) + (1-\lambda)F(w_1), \qquad \text{for any } \lambda \in [0,1].$$

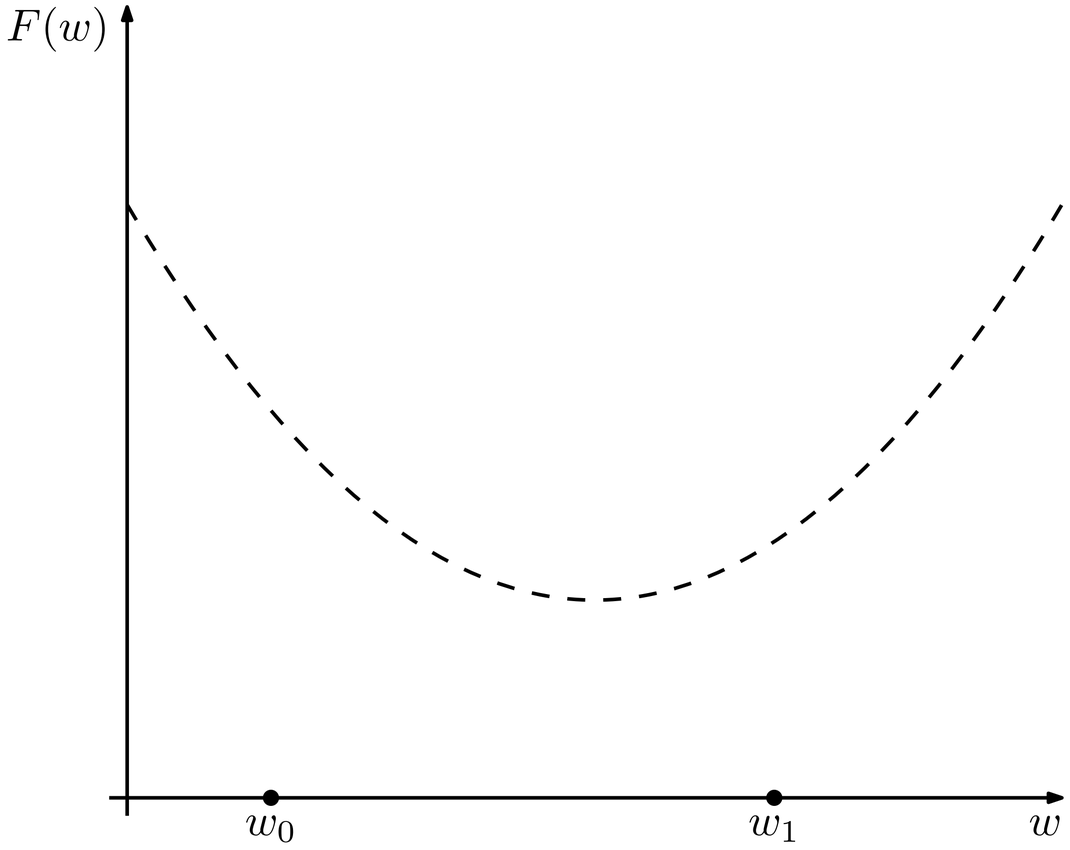
Convex function



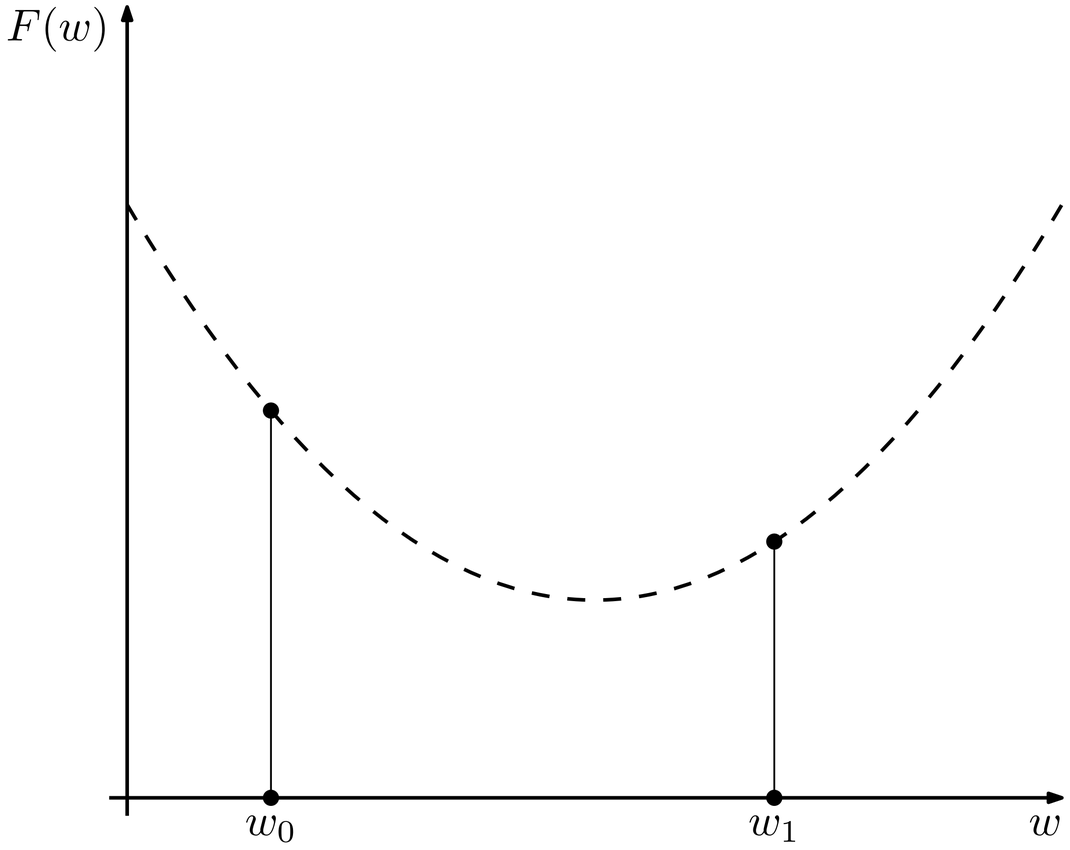
Convex function



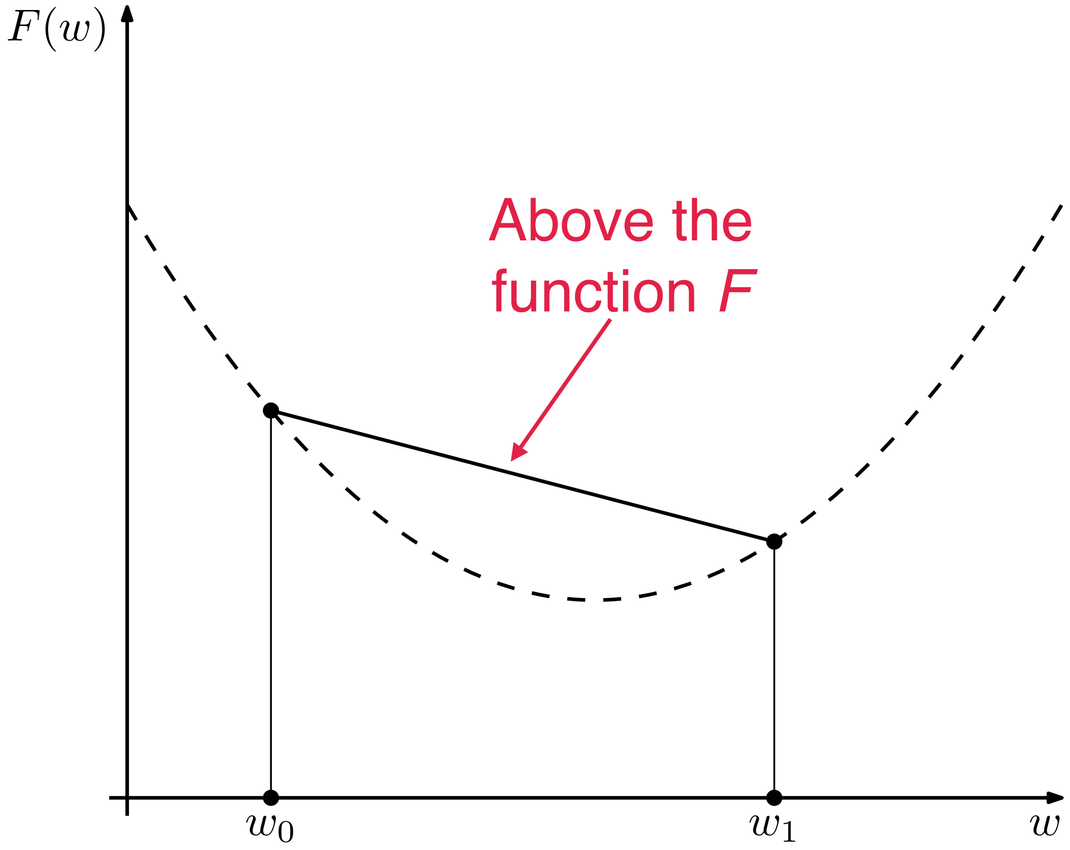
Convex function



Convex function



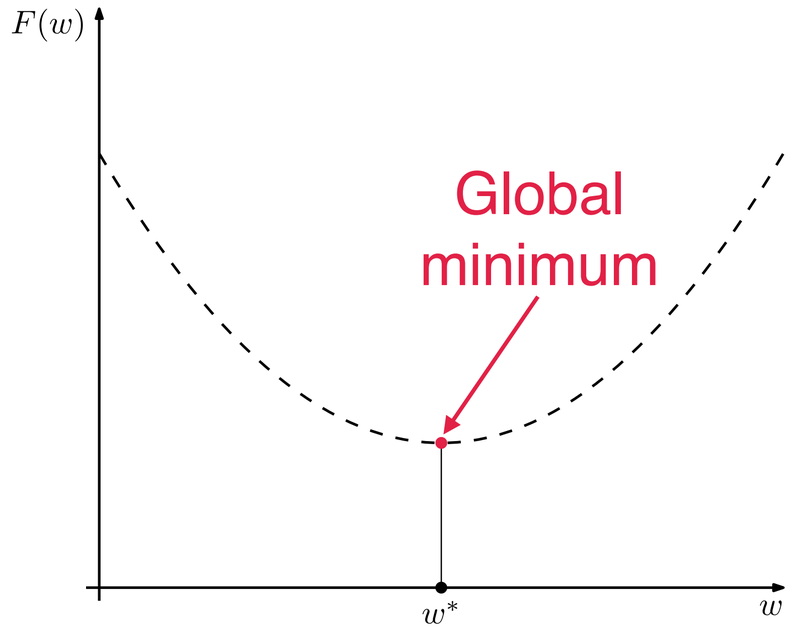
Convex function



TÉCNICO+
FORMAÇÃO AVANÇADA

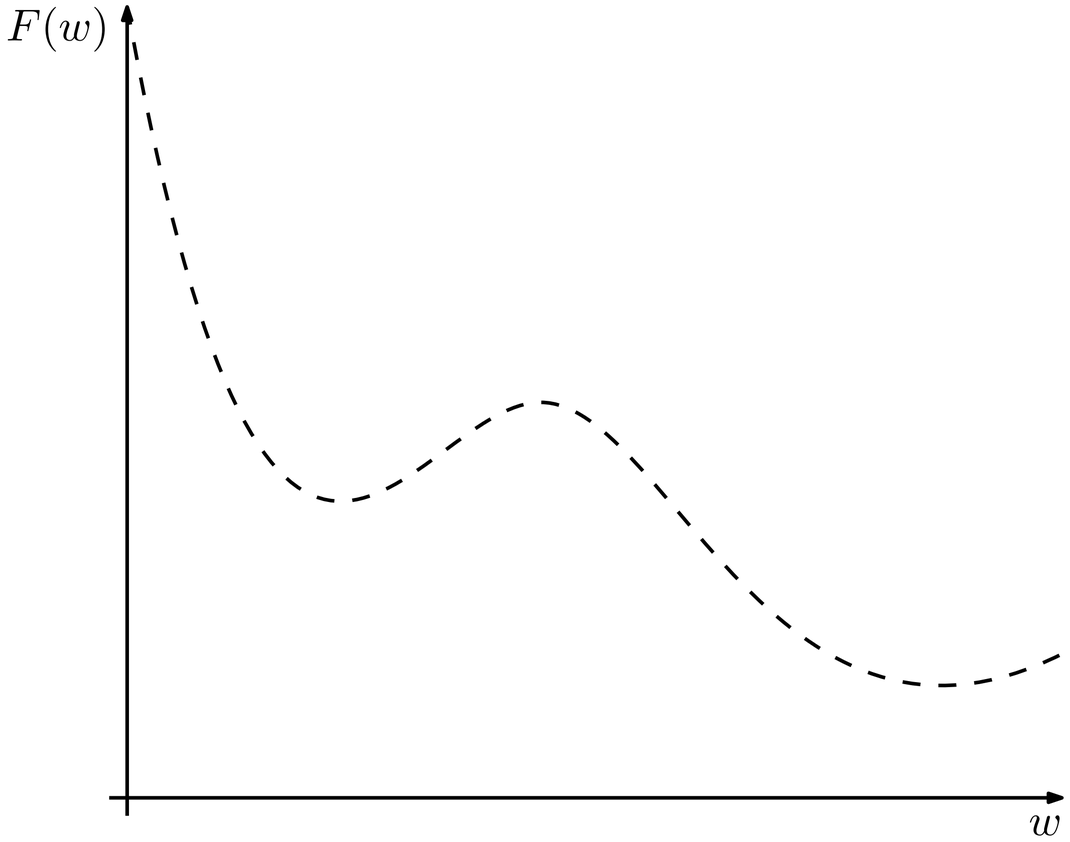
Local minima

- If the function is convex, perfect!
  - A point w^* where the gradient is zero is a global minimum

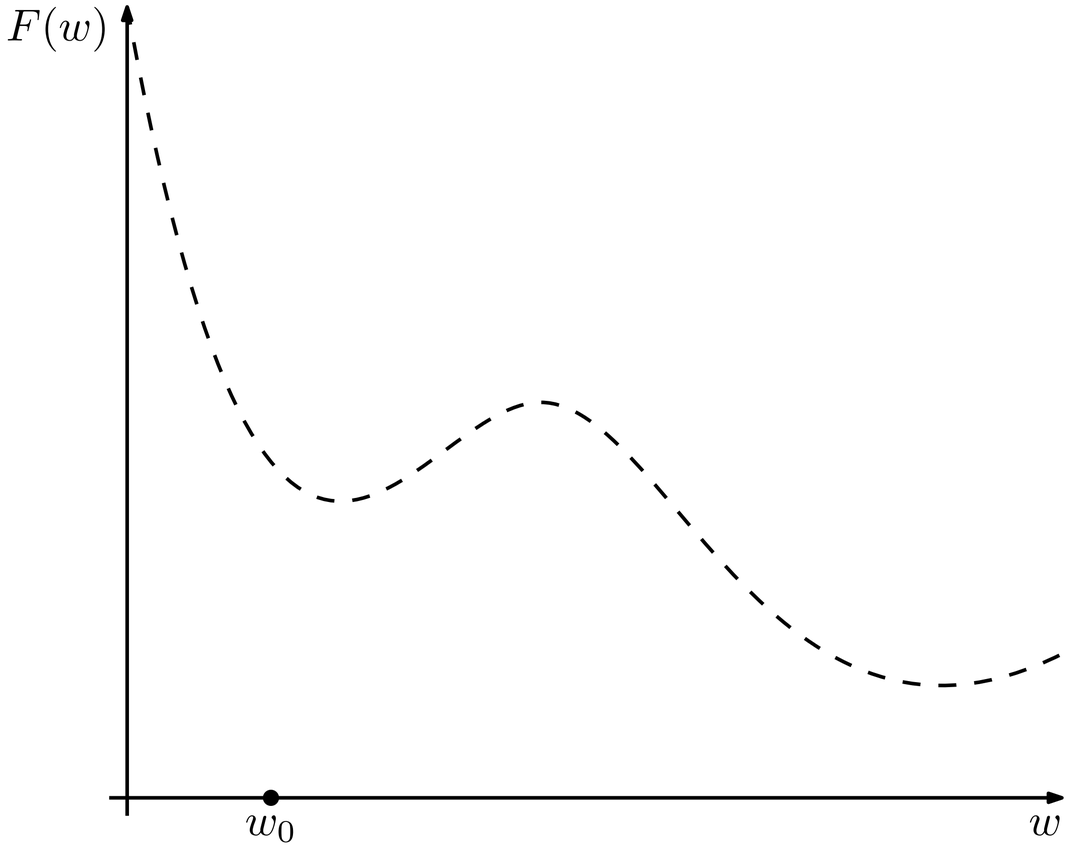


TÉCNICO+
FORMAÇÃO AVANÇADA

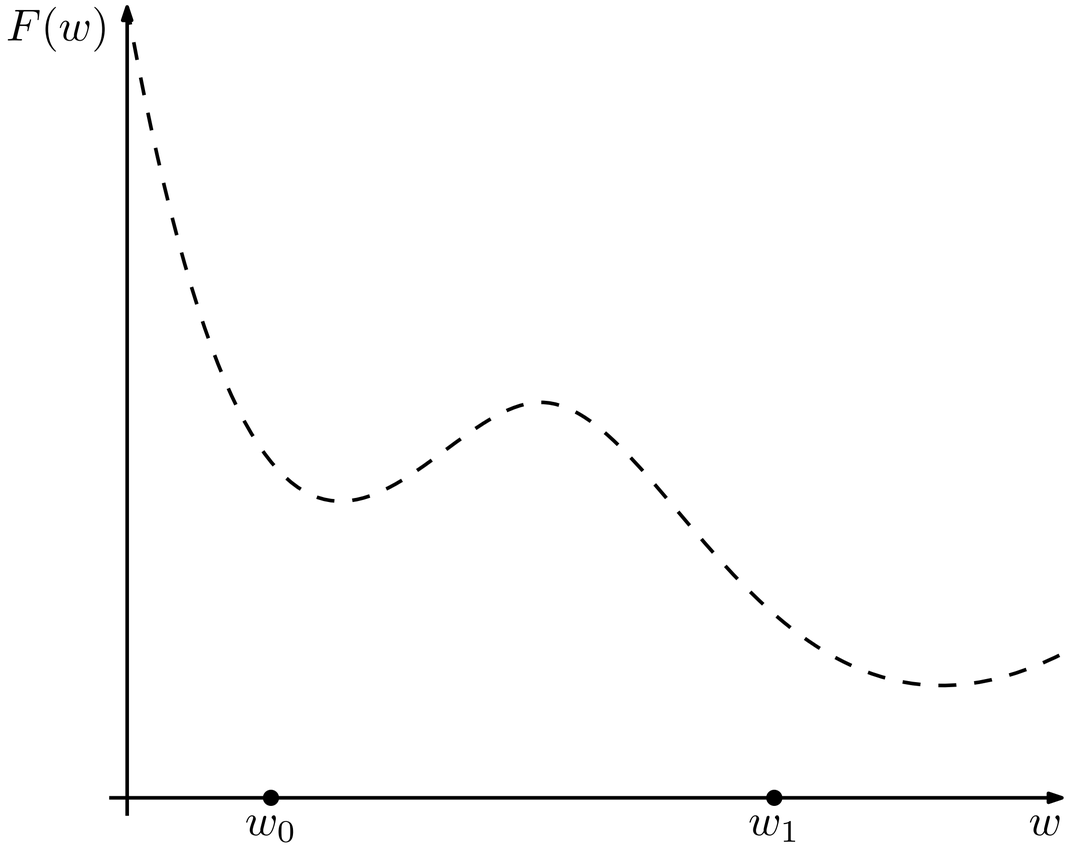
Non-convex function



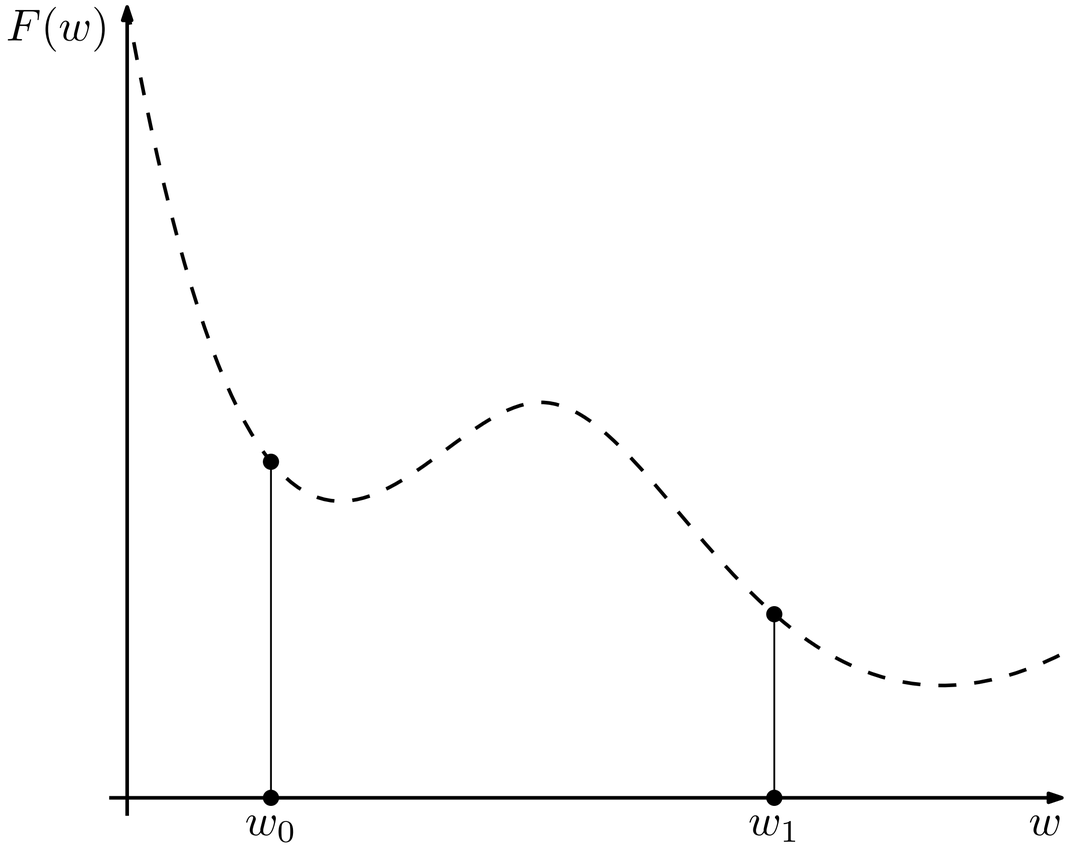
Non-convex function



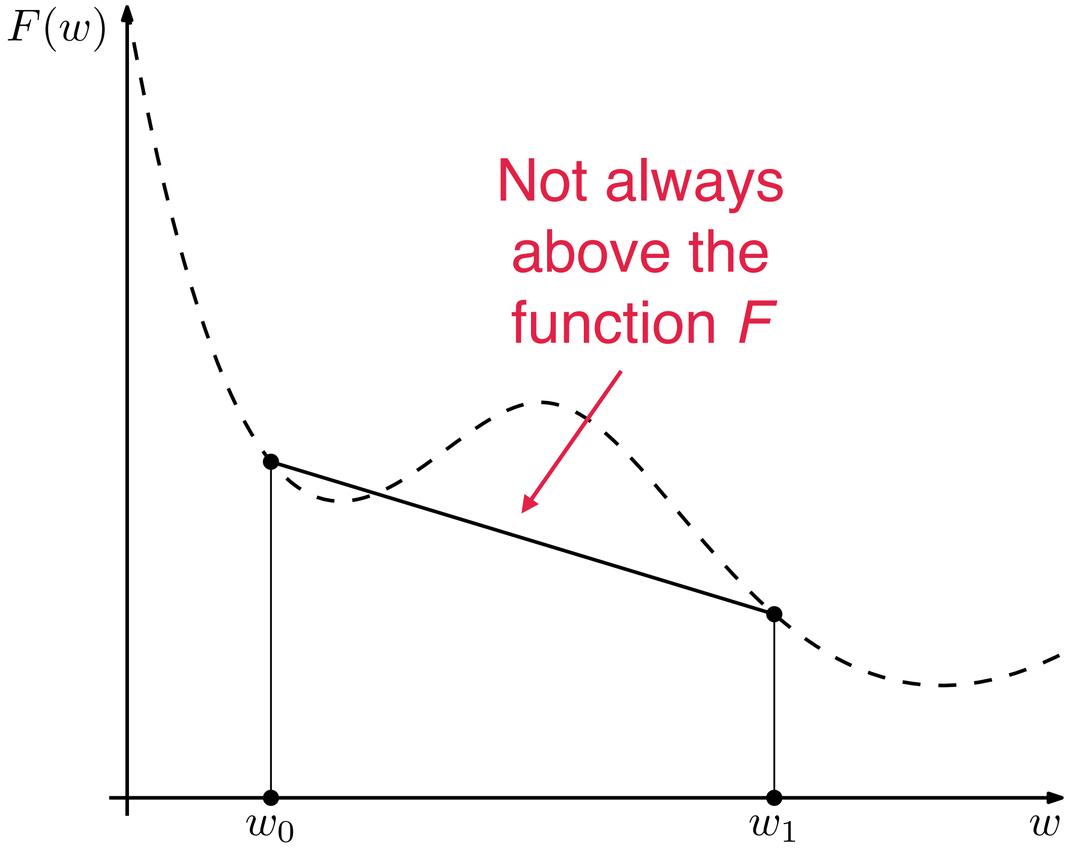
Non-convex function



Non-convex function

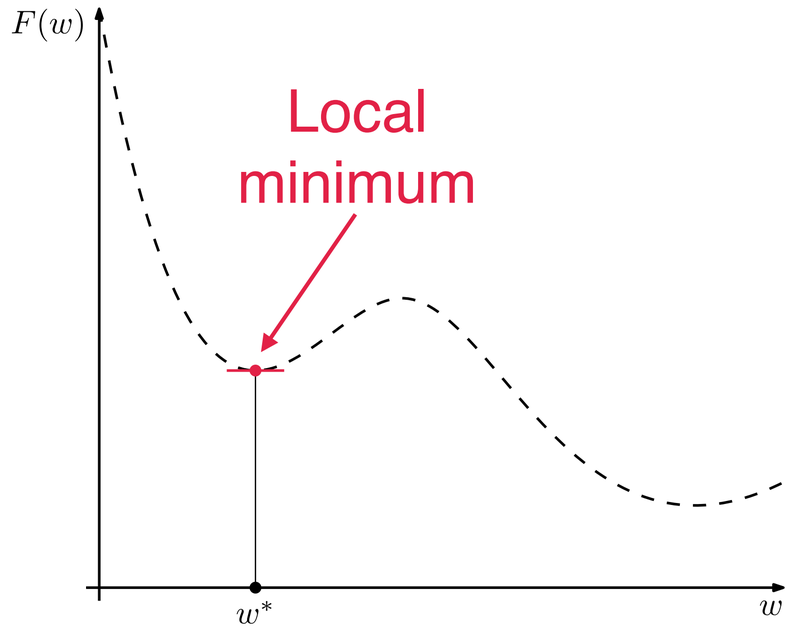


Non-convex function



Local minima

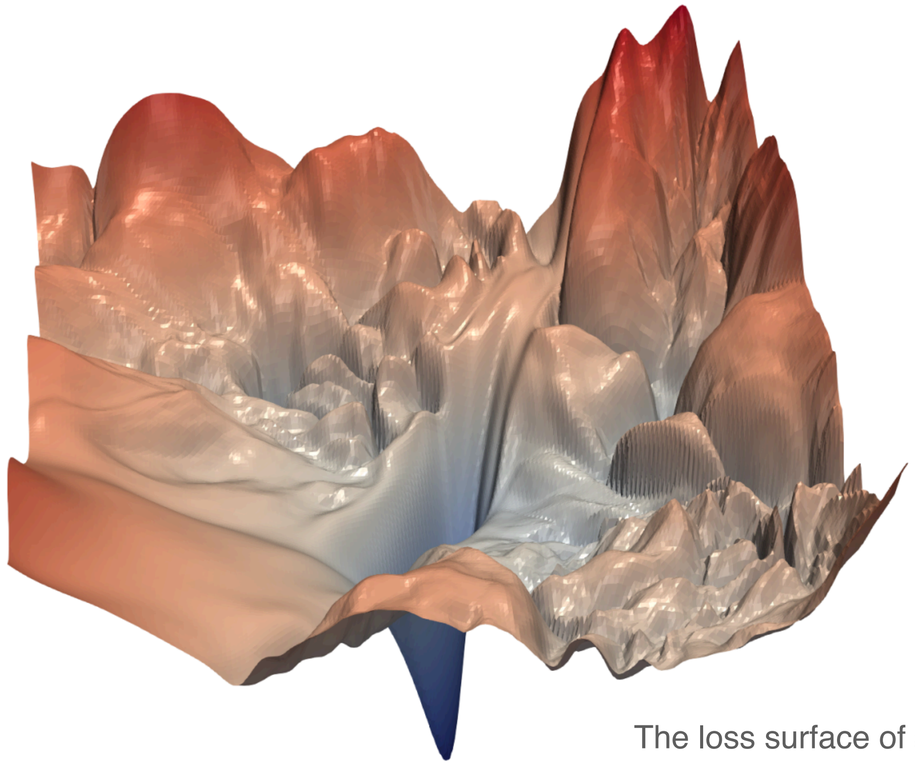
- If the function is non-convex, a point w^* where the gradient is zero may be a local minimum



TÉCNICO+
FORMAÇÃO AVANÇADA

What about neural networks?

- Is the loss of neural networks convex?



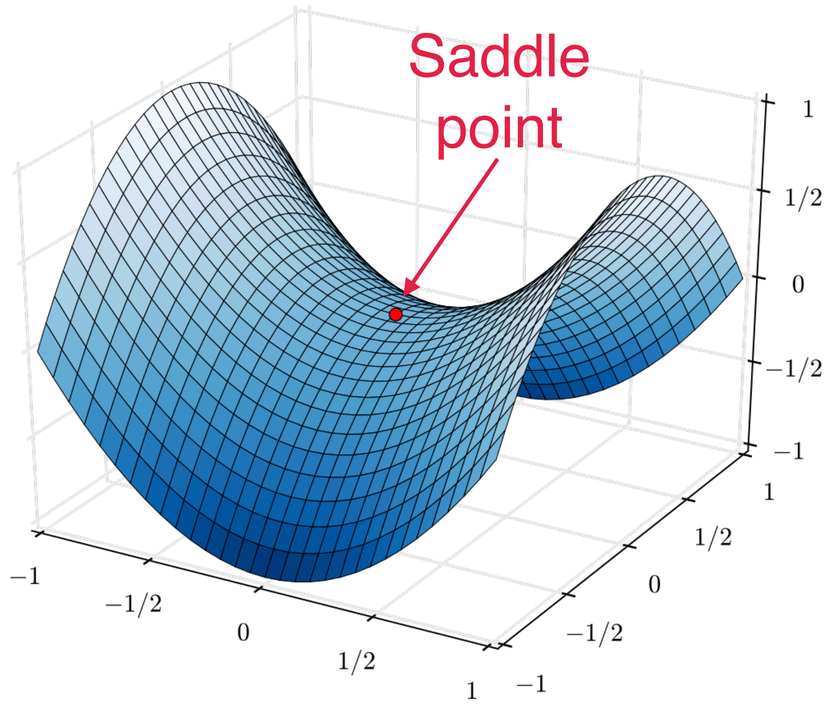
The loss surface of a ResNet-56.

Local minima

- Nevertheless, …
  - It is hard to assess whether a point is a local minimum or not
  - A common view among DL practitioners is that minima, even if local, are usually pretty good
  - Other optimization problems are significantly more serious

Saddle points

- A more serious problem is the existence of other points with null gradient, such as saddle points:



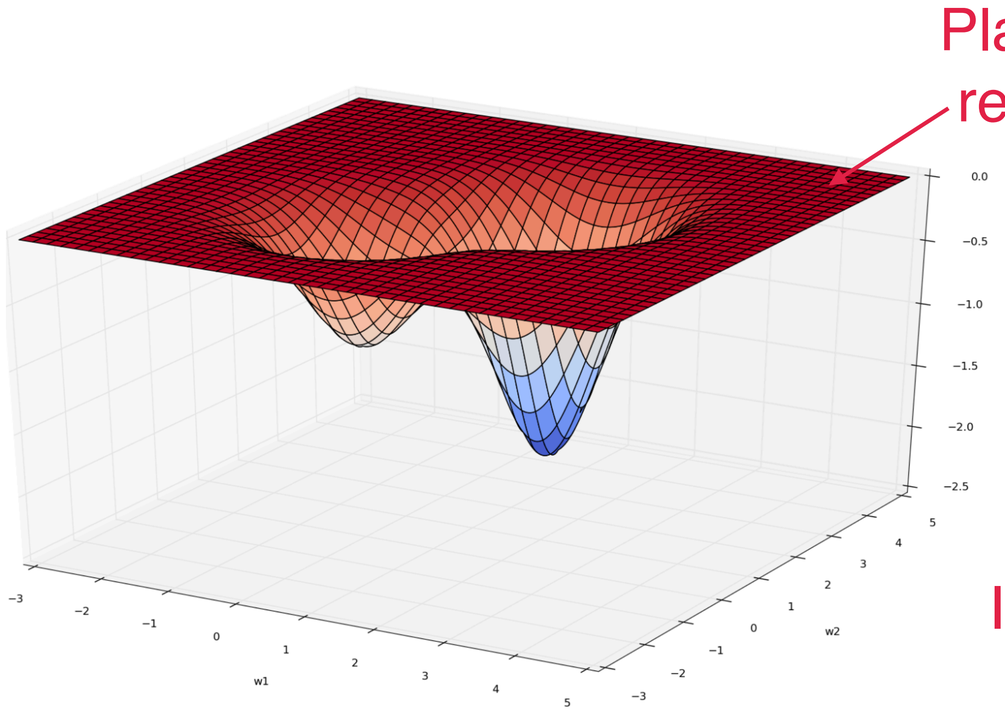
Saddle points are much more frequent than local minima/maxima

Saddle points

- A more serious problem is the existence of other points with null gradient, such as saddle points
  - If the algorithm lands on a saddle point, it will get “stuck”
  - Saddle points can be particularly harmful to second-order methods (e.g., Newton’s method)

Plateaux and flat regions

- Plateaux are large, flat regions of the optimization landscape

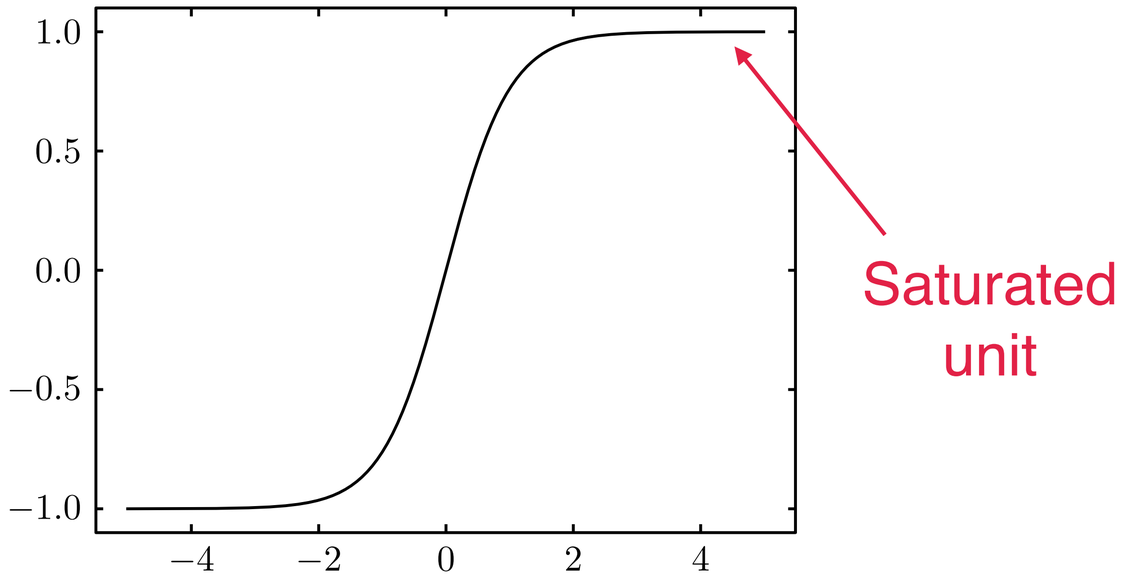


In plateaux (and other flat regions), the gradient is (approx.) zero, and learning makes little progress

TÉCNICO+
FORMAÇÃO AVANÇADA

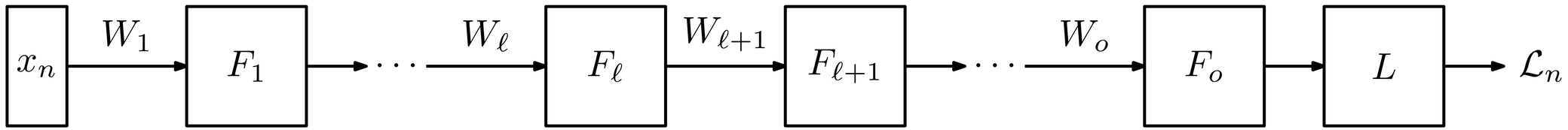
Plateaux and flat regions

- In plateaux, the value of the function can be very high
- Plateaux can happen, for example, if the units in a network are saturated



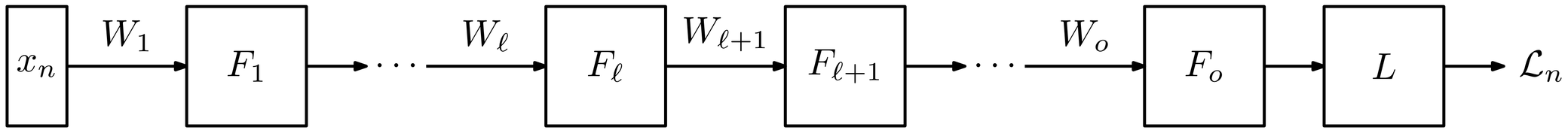
TÉCNICO+
FORMAÇÃO AVANÇADA

Vanishing and exploding gradients



- Recall, from the previous lecture, the back propagation computations
  - $\nabla_{W_o}\mathcal{L}_n = \nabla_{z_o}\mathcal{L}_n h_L^T,$ with $\nabla_{z_o}\mathcal{L}_n = \text{diff}(y(x_n), y_n)$
  - $\nabla_{W_L}\mathcal{L}_n = \nabla_{z_L}\mathcal{L}_n h_{L-1}^T,$ with $\nabla_{z_L}\mathcal{L}_n = \nabla_{h_L}\mathcal{L}_n \odot F_L'(z_L)$ and $\nabla_{h_L}\mathcal{L}_n = W_o^T \nabla_{z_o}\mathcal{L}_n$
  - ...
  - $\nabla_{W_1}\mathcal{L}_n = \nabla_{z_1}\mathcal{L}_n x,$ with $\nabla_{z_1}\mathcal{L}_n = \nabla_{h_1}\mathcal{L}_n \odot F_1'(z_1)$ and $\nabla_{h_1}\mathcal{L}_n = W_2^T \nabla_{z_2}\mathcal{L}_n$

Vanishing and exploding gradients



- If (for simplicity) all layers have linear activation,
  - $\nabla_{W_o}\mathcal{L}_n = \nabla_{z_o}\mathcal{L}_n h_L^T,$ with $\nabla_{z_o}\mathcal{L}_n = \text{diff}(y(x_n), y_n)$
  - $\nabla_{W_L}\mathcal{L}_n = \nabla_{z_L}\mathcal{L}_n h_{L-1}^T,$ with $\nabla_{z_L}\mathcal{L}_n = W_o^T \nabla_{z_o}\mathcal{L}_n$
  - ...
  - $\nabla_{W_1}\mathcal{L}_n = \nabla_{z_1}\mathcal{L}_n x,$ with $\nabla_{z_1}\mathcal{L}_n = W_2^T \nabla_{z_2}\mathcal{L}_n = \prod_{\ell=1}^{L} W_{\ell+1}^T \mathcal{L}_n$

TÉCNICO+
FORMAÇÃO AVANÇADA

Vanishing and exploding gradients

- Since

Product of many matrices

$$\nabla_{W_\ell} \mathcal{L}_n = \prod_{k=\ell}^{L} W_{k+1}^T \mathcal{L}_n$$

in deep networks, the gradient with respect to earlier units can be:

- Very small, if the weight matrices are small (vanishing gradients)
- Very large, if the weight matrices are large (exploding gradients)

Vanishing gradient

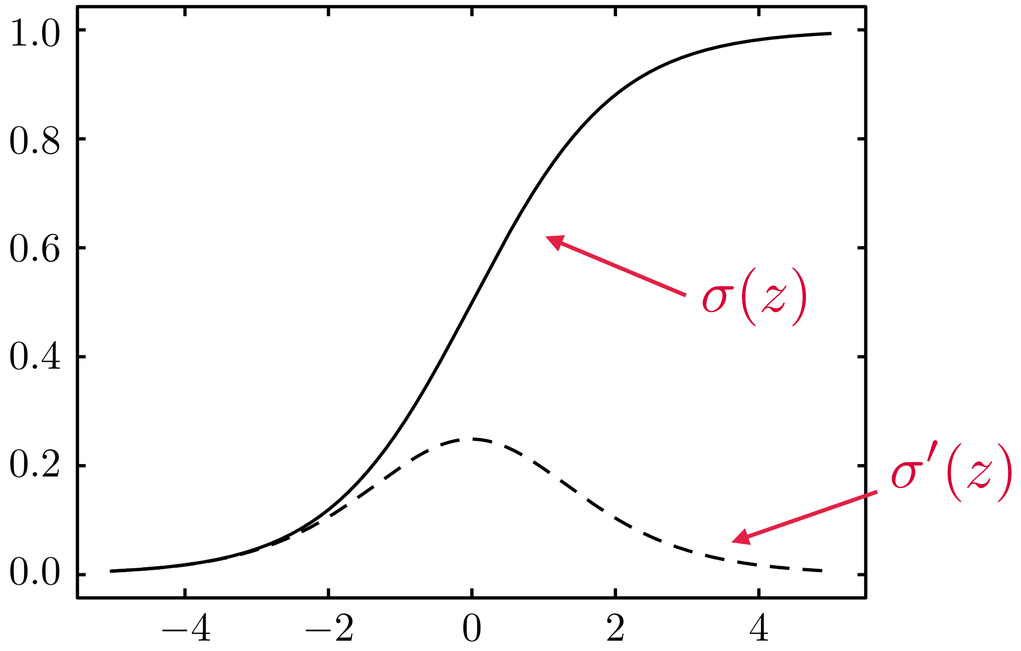
```python
import torch

# Let us create a 4 x 4 random matrix with entries drawn according to a
# Gaussian with zero mean and unit variance
M = torch.normal(0, 1, size=(4,4))
print('Initial matrix M:')
print(M)

# We now multiply out matrix 100 times by other similar random matrices
for i in range(100):
    M = torch.mm(M, torch.normal(0, 0.5, size=(4, 4)))

print('Matrix M after 100 multiplications:')
print(M)
```

```
Initial matrix M:
tensor([[ 0.2346,  0.7083, -1.5196, -0.2738],
        [ 1.0485,  0.0116,  1.1101, -0.0869],
        [ 0.7182,  2.4367, -0.0754, -0.6416],
        [-0.3200, -1.7107, -0.4235, -0.8431]])
Matrix M after 100 multiplications:
tensor([[ 3.1289e-07,  6.1478e-07, -3.4035e-07, -2.6486e-07],
        [-1.1187e-06, -2.1981e-06,  1.2169e-06,  9.4698e-07],
        [-1.5900e-06, -3.1242e-06,  1.7296e-06,  1.3460e-06],
        [ 1.5226e-06,  2.9917e-06, -1.6562e-06, -1.2889e-06]])
```

Note the size of these numbers!

Vanishing gradient

- Vanishing gradients are also due to the small gradient of certain activation functions (e.g., sigmoid or hyperbolic tangent)



Exploding gradients

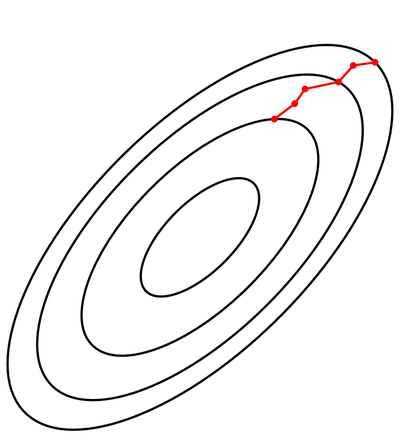
```
import torch

# Let us create a 4 x 4 random matrix with entries drawn according to a
# Gaussian with zero mean and unit variance
M = torch.normal(0, 1, size=(4,4))
print('Initial matrix M:')
print(M)

# We now multiply out matrix 100 times by other similar random matrices
for i in range(100):
    M = torch.mm(M, torch.normal(0, 1, size=(4, 4)))

print('Matrix M after 100 multiplications:')
print(M)
```

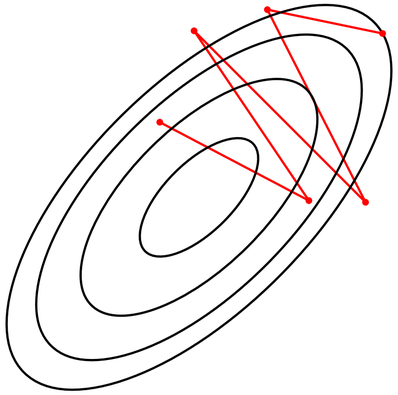
```
Initial matrix M:
tensor([[ 0.7277, -1.0278, -1.2209, -1.0516],
        [-2.0197, -0.5839, -0.6235,  0.3025],
        [ 0.3636,  0.2927,  0.7657,  1.5011],
        [ 0.8449,  0.9412, -0.2234,  0.1971]])
Matrix M after 100 multiplications:
tensor([[-3.1648e+22, -5.2418e+22,  5.3503e+22, -2.4471e+22],
        [ 1.8854e+22,  3.1227e+22, -3.1874e+22,  1.4578e+22],
        [ 1.6210e+22,  2.6849e+22, -2.7405e+22,  1.2534e+22],
        [-9.4666e+21, -1.5679e+22,  1.6004e+22, -7.3199e+21]])
```

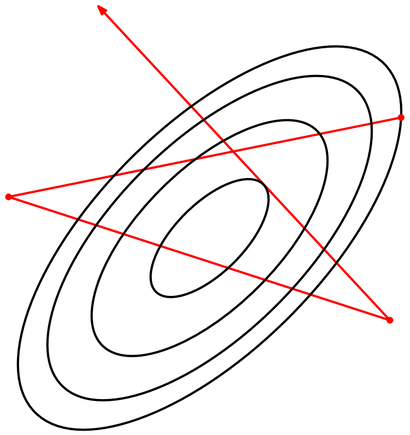
Note the size of these numbers!

Inexact gradients

- Using mini-batch/stochastic gradient descent, gradient direction is estimated based on samples
  - Noisy gradient estimates combined with large step-size choice may lead to oscillation/divergence







Going beyond SGD…

Mini-batch gradient descent

- We use mini-batches to avoid:
  - The computational burden of computing full gradients
  - The variance in stochastic gradient descent
- We have the update rule

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \alpha \sum_{n \in \mathcal{B}_k} \nabla_{W_\ell} \underbrace{(L(x_n, y_n; W) + \lambda R(W))}_{\mathcal{L}_n(W)}$$

Mini-batch gradient descent

- We use mini-batches to avoid:
  - The computational burden of computing full gradients
  - The variance in stochastic gradient descent
- We get the update rule

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \alpha \underbrace{\sum_{n \in \mathcal{B}_k} \nabla_{W_\ell} \mathcal{L}_n(W)}_{g_k}$$

Gradient of batch *k*

TÉCNICO+
FORMAÇÃO AVANÇADA

Mini-batch gradient descent

- We use mini-batches to avoid:
  - The computational burden of computing full gradients
  - The variance in stochastic gradient descent
- We get the update rule

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \alpha g_k$$

Mini-batch gradient descent

- We use mini-batches to avoid:
  - The computational burden of computing full gradients
  - The variance in stochastic gradient descent
- We get the update rule

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \alpha g_k$$

$$g_k = \sum_{n \in \mathcal{B}_k} \nabla_{W_\ell} \mathcal{L}_n(W)$$

TÉCNICO+
FORMAÇÃO AVANÇADA

Momentum

- Can we further decrease the variance of the gradient estimate?
- Note that

$$\sum_{n=1}^{N} \nabla_W \mathcal{L}_n(W) = \sum_k g_k$$

- We can keep track of past gradients and average them together:

$$v_{k+1} \leftarrow \beta v_k + g_k$$

Improved gradient estimate

Forgetting factor

Gradient of batch *k*

Momentum

- The resulting method is known as mini-batch/stochastic gradient descent with momentum
- Update rule:

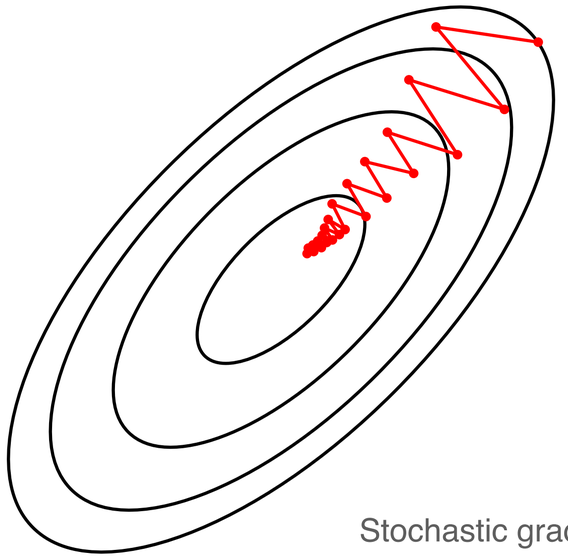
$$v_{k+1} \leftarrow \beta v_k + g_k$$

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \alpha v_{k+1}$$

Uses improved estimate to update

Momentum

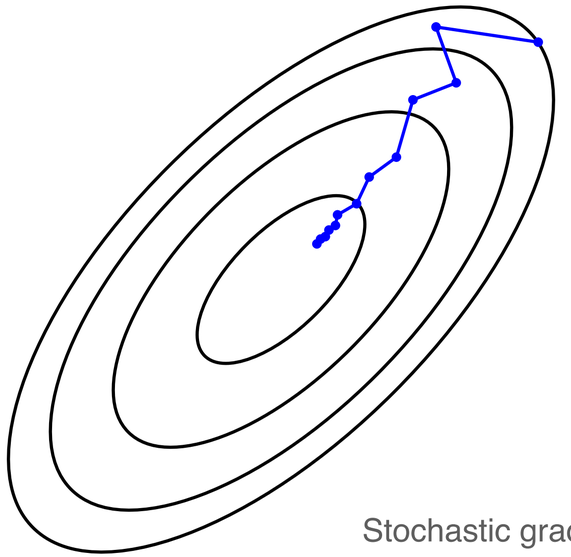
- Intuitively, the update gains “momentum” from previous displacements
- Ends up reinforcement main descent direction and ignoring noise



Stochastic gradient descent

Momentum

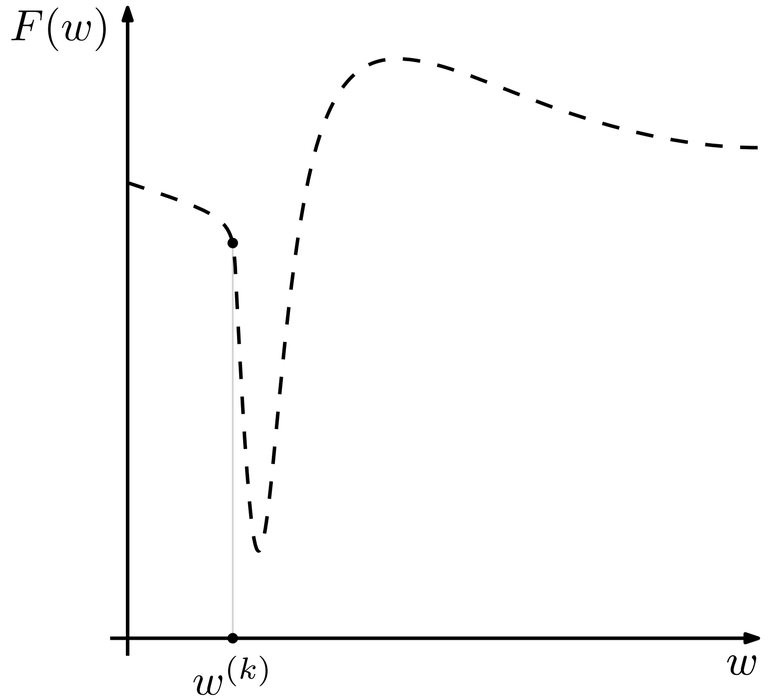
- Intuitively, the update gains “momentum” from previous displacements
- Ends up reinforcement main descent direction and ignoring noise



Stochastic gradient descent with momentum

Adaptive step-sizes

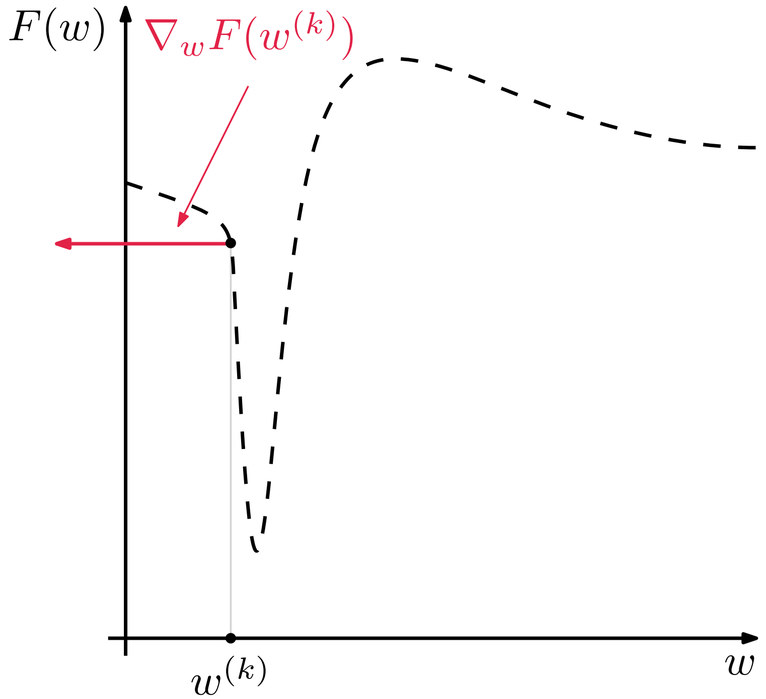
- Depending on the slope of the loss function, we may need to use different step-sizes…



TÉCNICO+
FORMAÇÃO AVANÇADA

Adaptive step-sizes

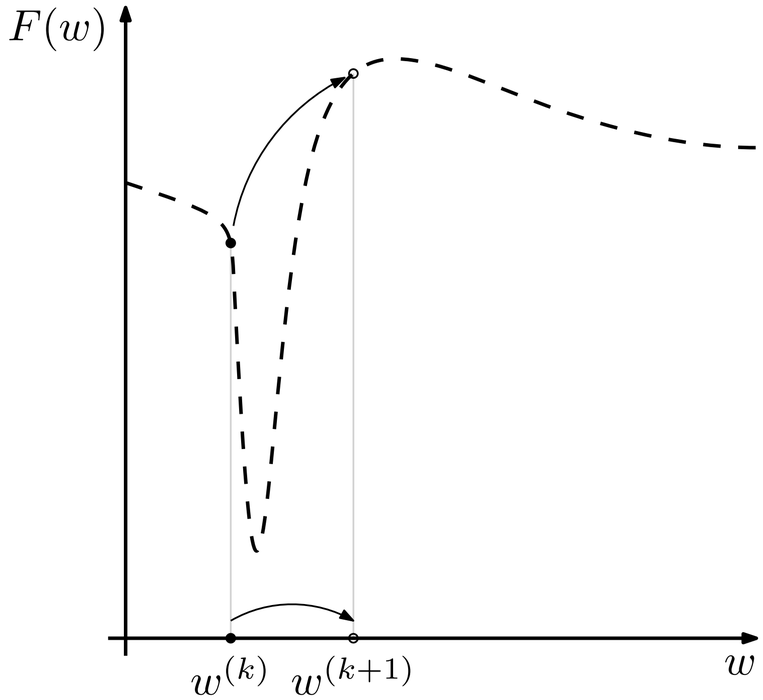
- Depending on the slope of the loss function, we may need to use different step-sizes…



TÉCNICO+
FORMAÇÃO AVANÇADA

Adaptive step-sizes

- Depending on the slope of the loss function, we may need to use different step-sizes…

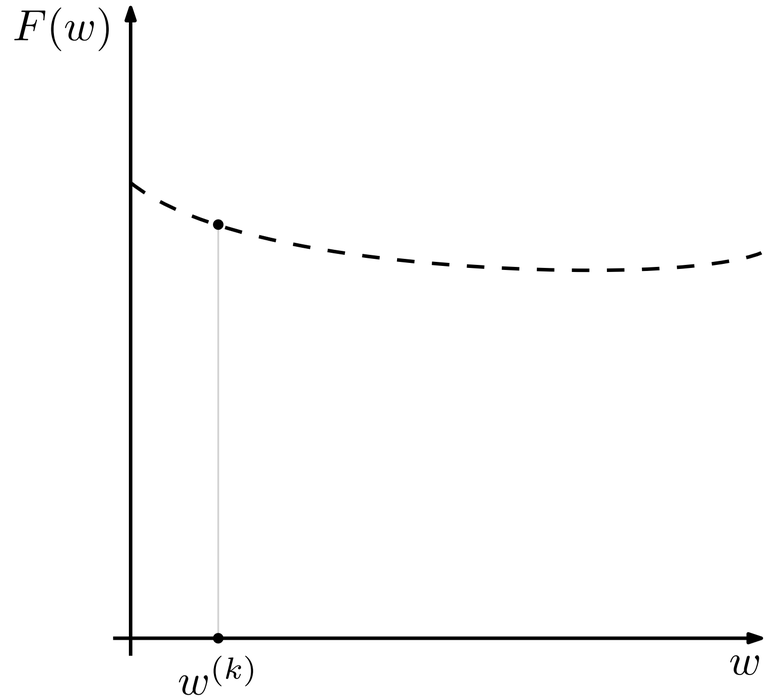


A small step size is necessary to avoid jumping over the minimum

TÉCNICO+
FORMAÇÃO AVANÇADA

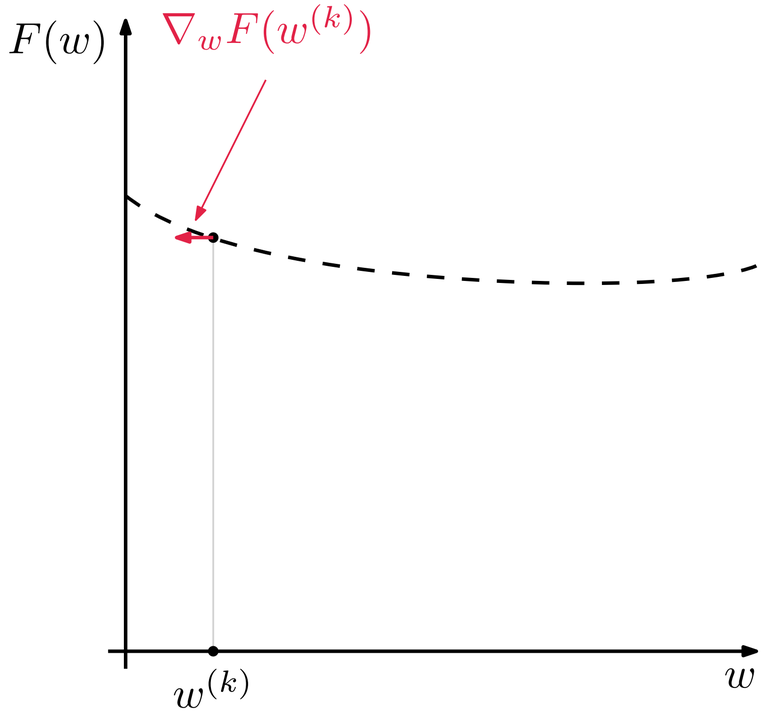
Adaptive step-sizes

- Depending on the slope of the loss function, we may need to use different step-sizes…



Adaptive step-sizes

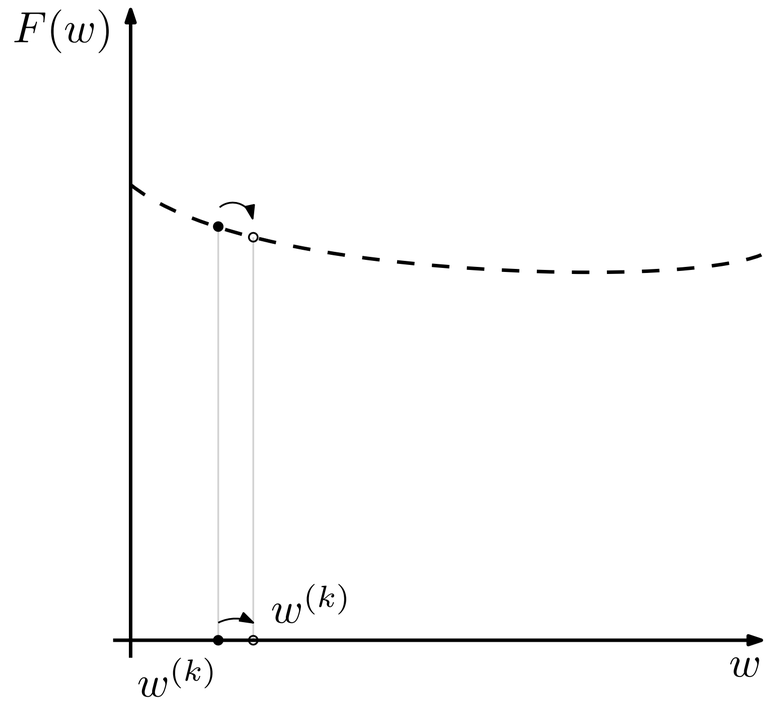
- Depending on the slope of the loss function, we may need to use different step-sizes…



TÉCNICO+
FORMAÇÃO AVANÇADA

Adaptive step-sizes

- Depending on the slope of the loss function, we may need to use different step-sizes…



A larger step size is necessary to attain faster convergence

TÉCNICO+
FORMAÇÃO AVANÇADA

Adaptive step-sizes

- Depending on the slope of the loss function, we may need to use different step-sizes
- Different components of the gradient may have different magnitudes

Adaptive step-sizes

- We can use a simple heuristic to address both problems:
  - We use a different step-size for each component
  - For each component, the step-size should be…
    - … small, if the gradient is large (in absolute value)
    - … large, if the gradient is small (in absolute value)

Adaptive step-sizes

- We could then set

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \boxed{\frac{\alpha}{\sqrt{g_k^2 + \epsilon}}} g_k$$

New step-size (division taken component-wise)

AdaGrad

- However, to be more robust to abrupt changes in step-size, we keep track of past gradients:

$$u_{k+1} \leftarrow u_k + g_k^2$$

resulting in the AdaGrad update

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \boxed{\frac{\alpha}{\sqrt{u_{k+1} + \epsilon}}} g_k$$

New step-size (division taken component-wise)

RMSProp

- A problem with AdaGrad is that u_k keeps growing without bound, making the algorithm eventually too slow (step-size too small)
- Instead, RMSProp uses the step-size update

$$u_{k+1} \leftarrow \beta u_k + (1-\beta) g_k^2$$

- The parameter update remains unchanged:

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \frac{\alpha}{\sqrt{u_{k+1} + \epsilon}} g_k$$

Adam

- Adam combines momentum with adaptive step-sizes
- We set

$$v_{k+1} \leftarrow \beta_1 v_k + (1-\beta_1) g_k$$

$$u_{k+1} \leftarrow \beta_2 u_k + (1-\beta_2) g_k^2$$

Momentum

Adaptive step-size

- The two constants are set so that

$$\beta_1 < \beta_2$$

Adam

- To avoid very small values of u_k and v_k in early updates, both are normalized as

$$\hat{v}_{k+1} = \frac{v_{k+1}}{1-\beta_1^k}$$

$$\hat{u}_{k+1} = \frac{u_{k+1}}{1-\beta_2^k}$$

- The update finally comes

$$W_\ell^{k+1} \leftarrow W_\ell^{(k)} - \frac{\alpha}{\sqrt{\hat{u}_{k+1}+\epsilon}}\hat{v}_{k+1}$$

Convolutional neural networks

What is a CNN?

- A convolutional neural network is a feedforward neural network with a specialized connectivity structure
  - CNNs are at the genesis of the DL revolution
  - Particularly suitable for data with spacial structure (e.g., images)

Model for the visual system

- In 1959, two researchers from Harvard (Hubel and Wiesel) proposed a hierarchical model for the visual system

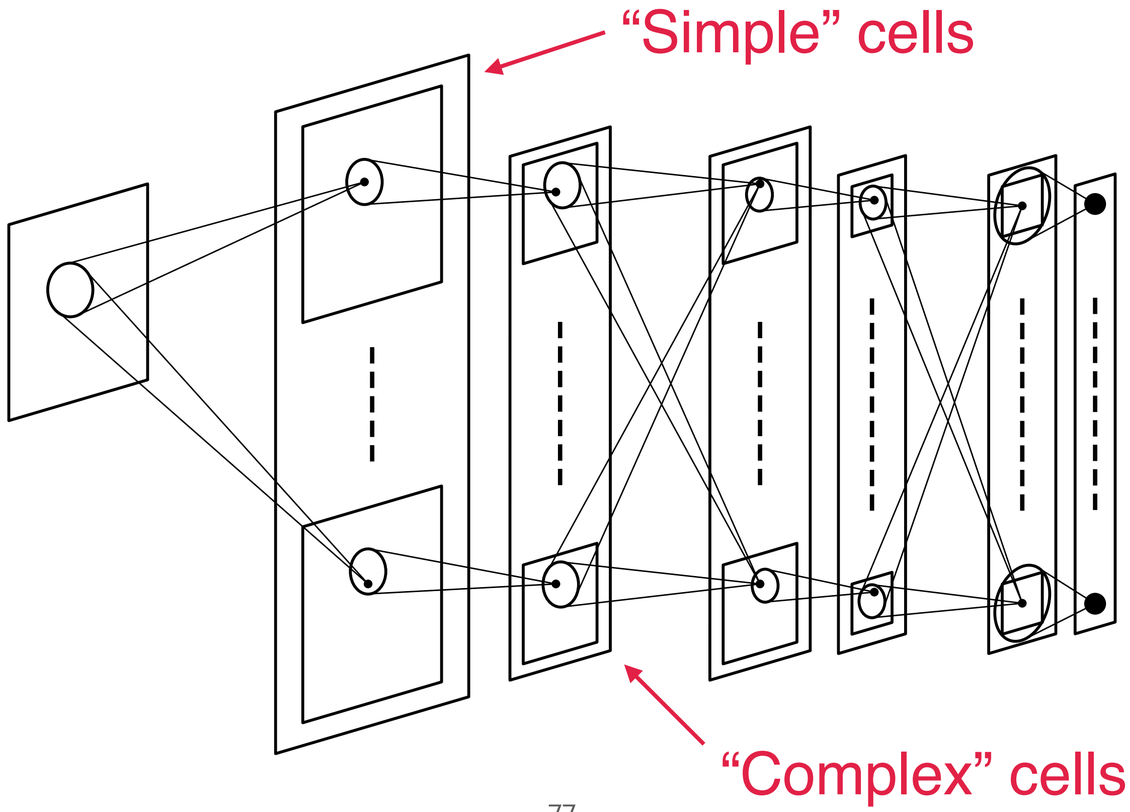
Hubel and Wiesel in Harvard University

Model for the visual system

- In 1959, two researchers from Harvard (Hubel and Wiesel) proposed a hierarchical model for the visual system
- In their model,
  - “Simple receptive fields”, arranged in linear structures, respond to simple patterns in light stimuli
  - “Complex receptive fields” respond to complex aggregations of patterns

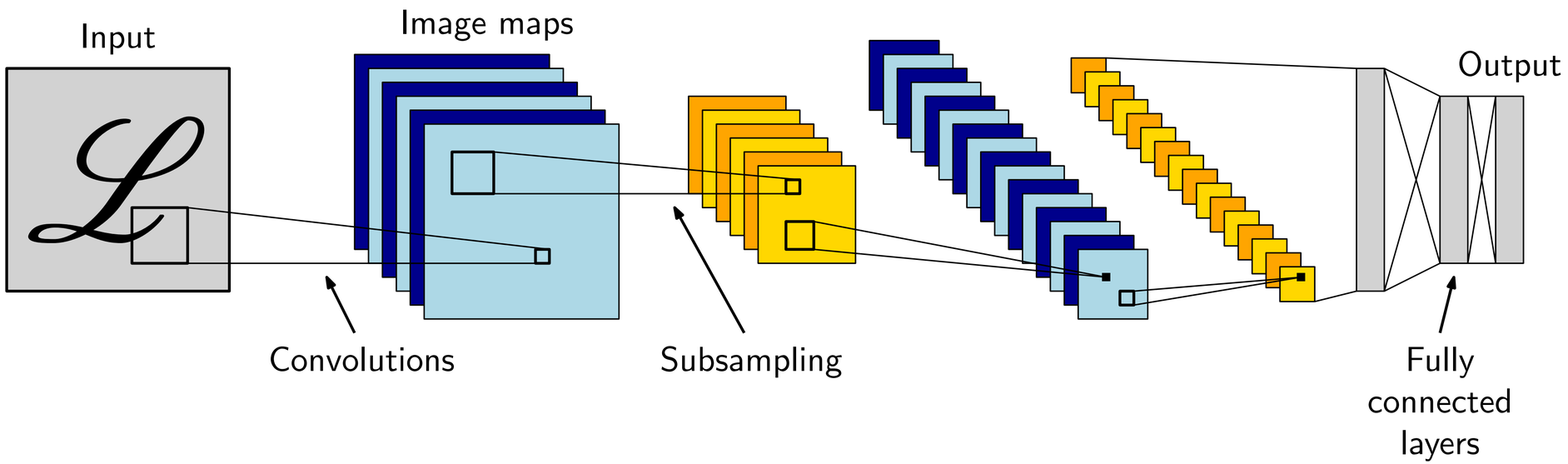
Neocognitron (1979)

- Inspired by the model of Hubel and Wiesel, Kunihiko Fukushima proposed in 1979 the Neocognitron



LeNet (1998)

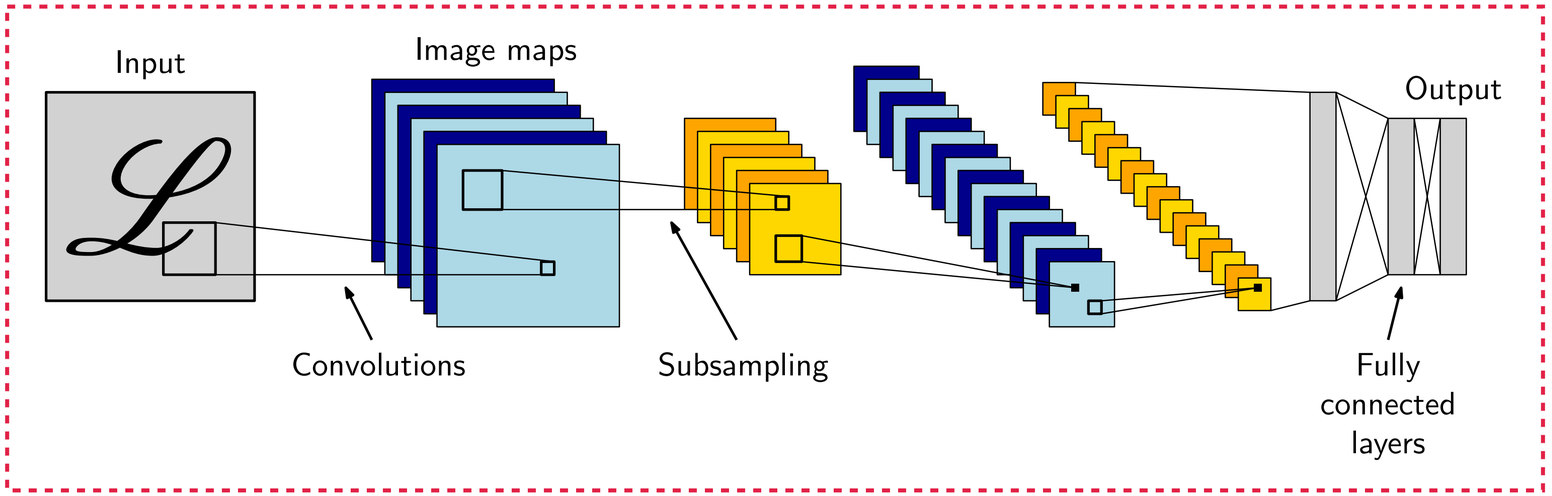
- The first convolutional neural network was proposed by LeCun et al in 1998

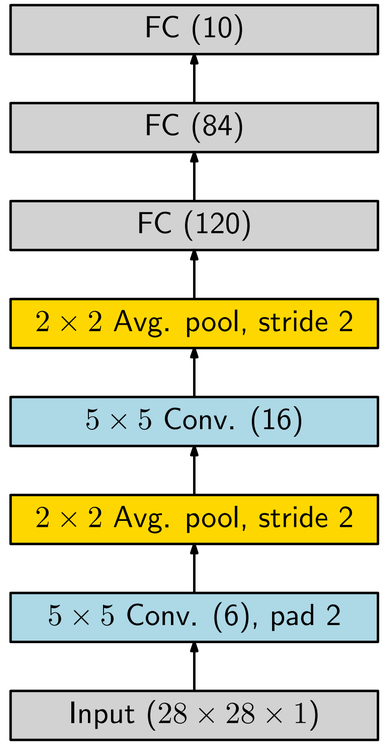


- Much smaller than current networks, but the architecture was similar to current CNNs

LeNet (1998)

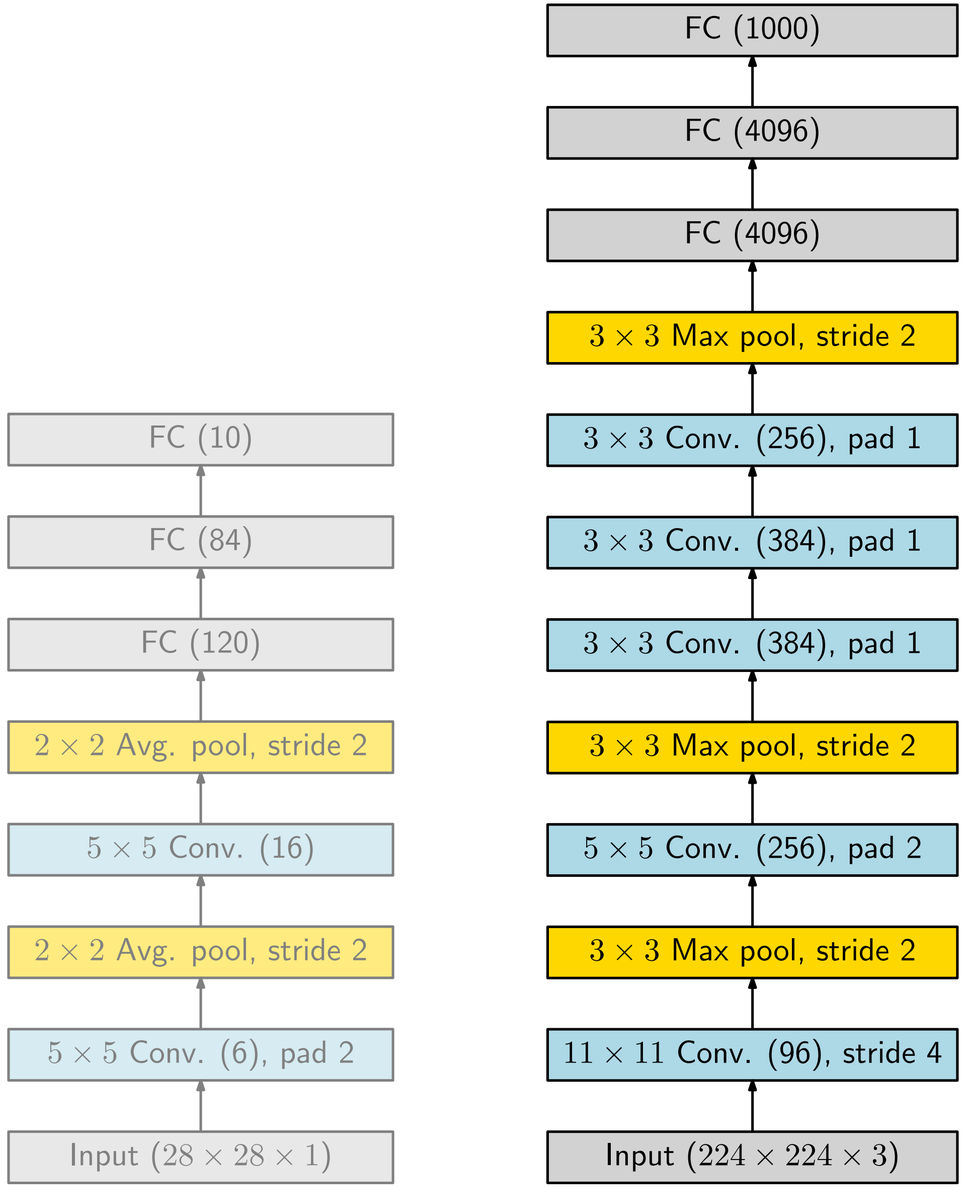
- More compactly…





AlexNet (2012)

- In 2012, AlexNet won the ImageNet Large Scale Visual Recognition Challenge, kicking off the “deep learning revolution”



TÉCNICO+
FORMAÇÃO AVANÇADA

Why convolutions?

- Desiderata for image processing tasks (e.g., face recognition):
  - We would like our system to be able to perform its task, even if the input image is shifted (invariance)

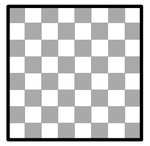
Can you find Waldo?

Why convolutions?

- Desiderata for image processing tasks (e.g., face recognition):
  - We would like our system to be able to perform its task, even if the input image is shifted (invariance)
  - Early layers in the network should focus on local features of the image, not the image as a whole (locality)

Why convolutions?

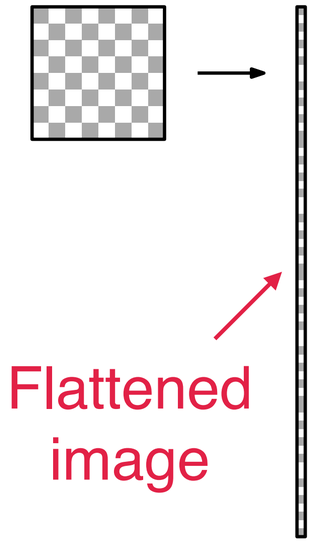
- Standard networks don’t meet the previous desiderata



Input image

Why convolutions?

- Standard networks don’t meet the previous desiderata



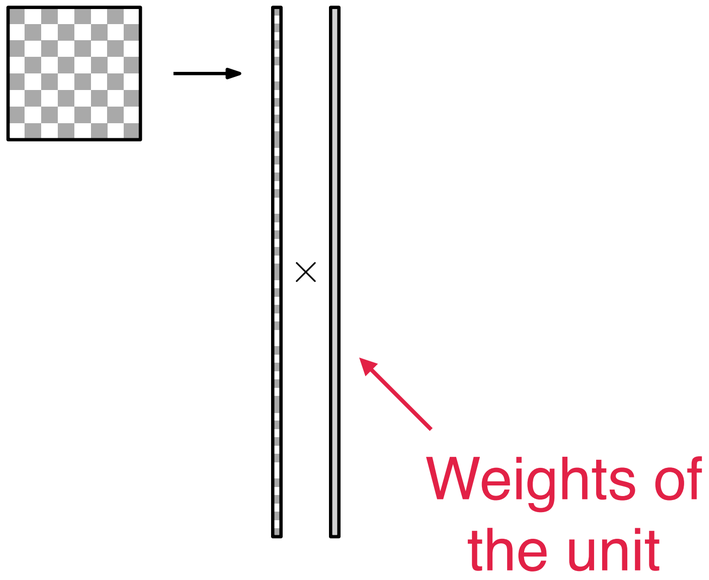
In flattening the image, we lose important spatial information

Why convolutions?

Does this look like a checkered pattern to you?

Why convolutions?

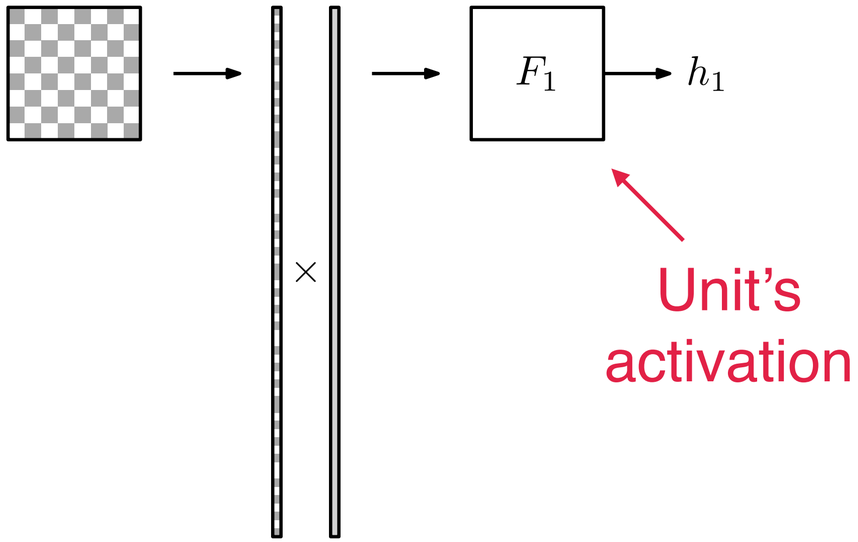
- Standard networks don’t meet the previous desiderata



Each unit is looking at the whole image (no locality)

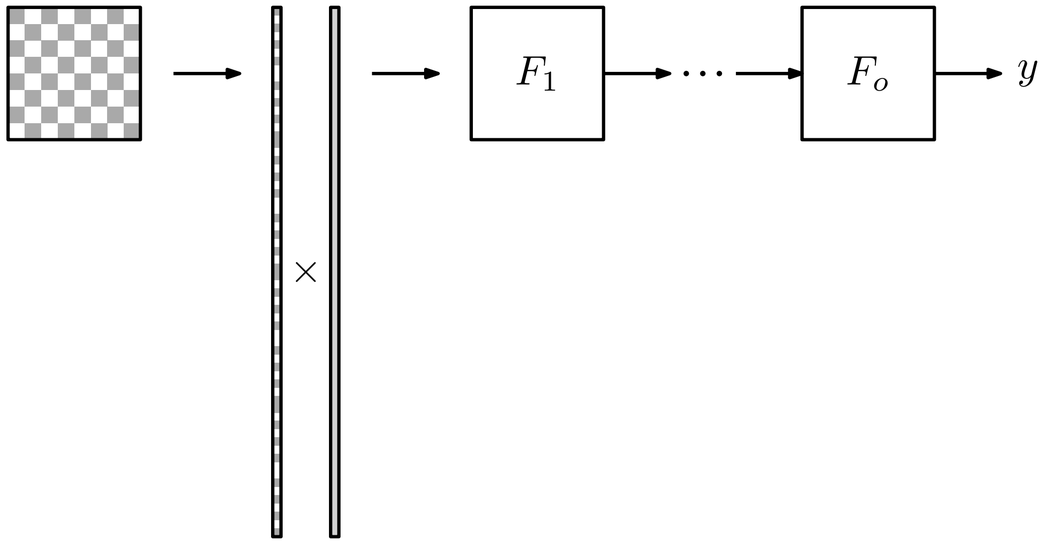
Why convolutions?

- Standard networks don’t meet the previous desiderata



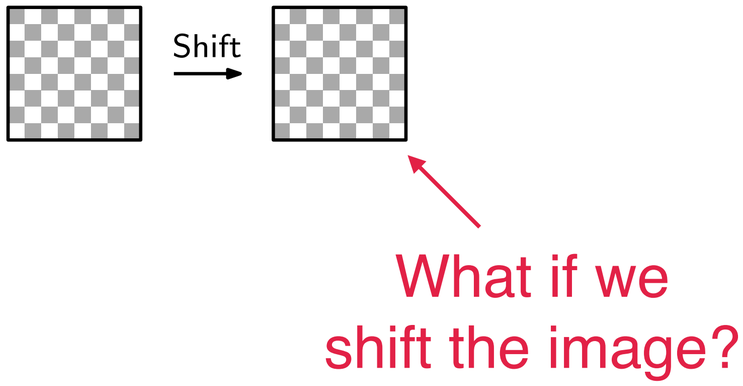
Why convolutions?

- Standard networks don’t meet the previous desiderata



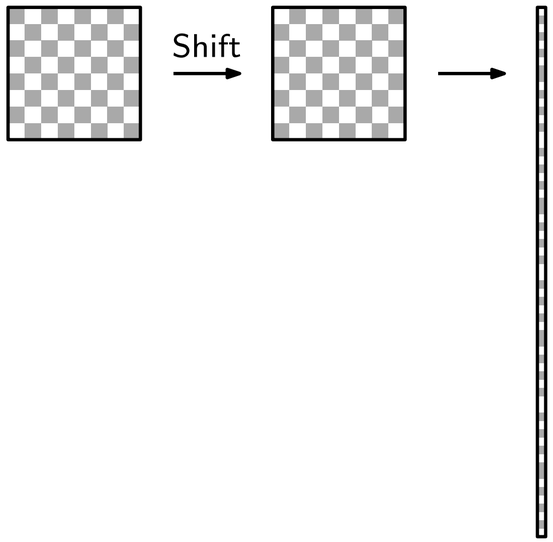
Why convolutions?

- Standard networks don’t meet the previous desiderata



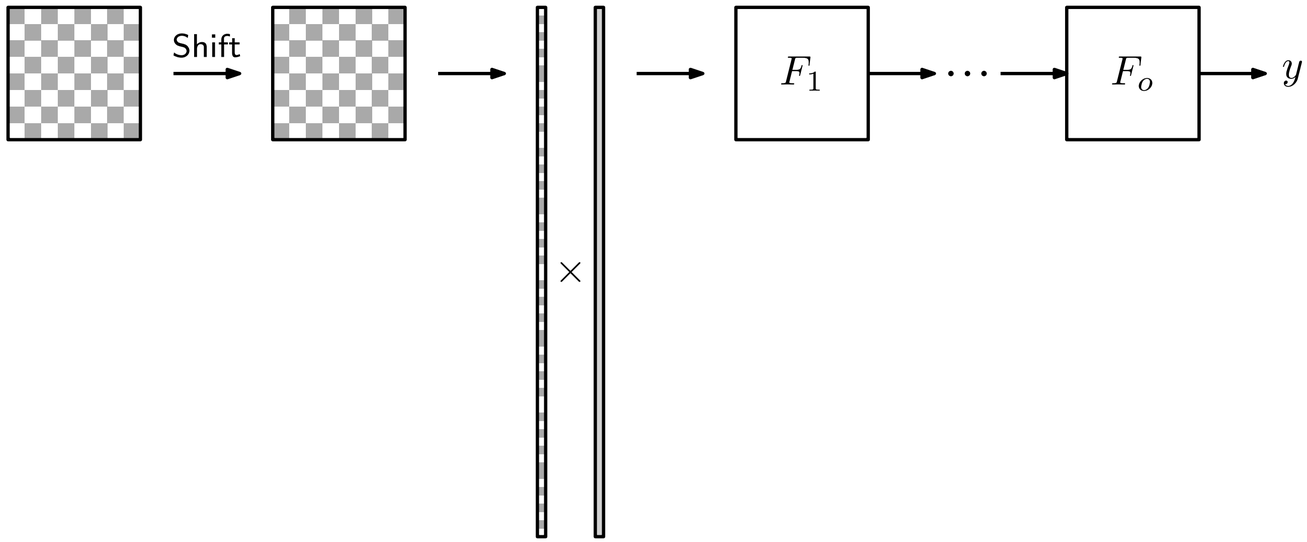
Why convolutions?

- Standard networks don’t meet the previous desiderata



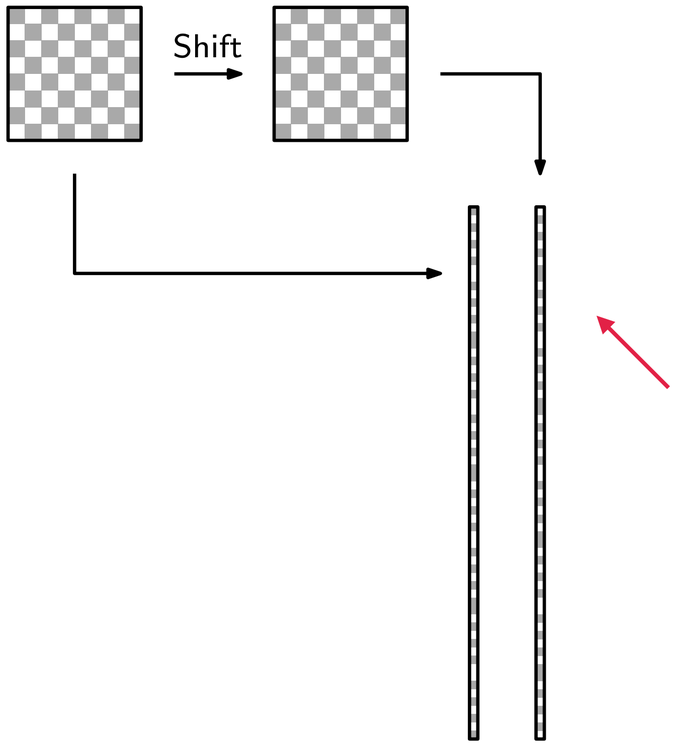
Why convolutions?

- Standard networks don’t meet the previous desiderata



Why convolutions?

- Standard networks don’t meet the previous desiderata



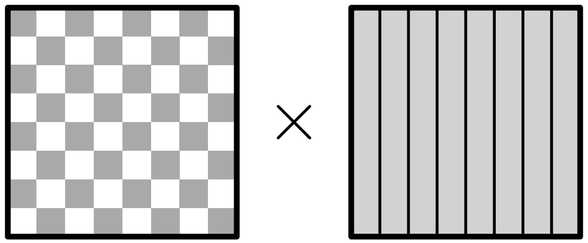
The two vectors are completely different

Sensitive to shifts in the image (no invariance)

Why convolutions?

- How can we change this?

    1. We maintain the spacial structure:

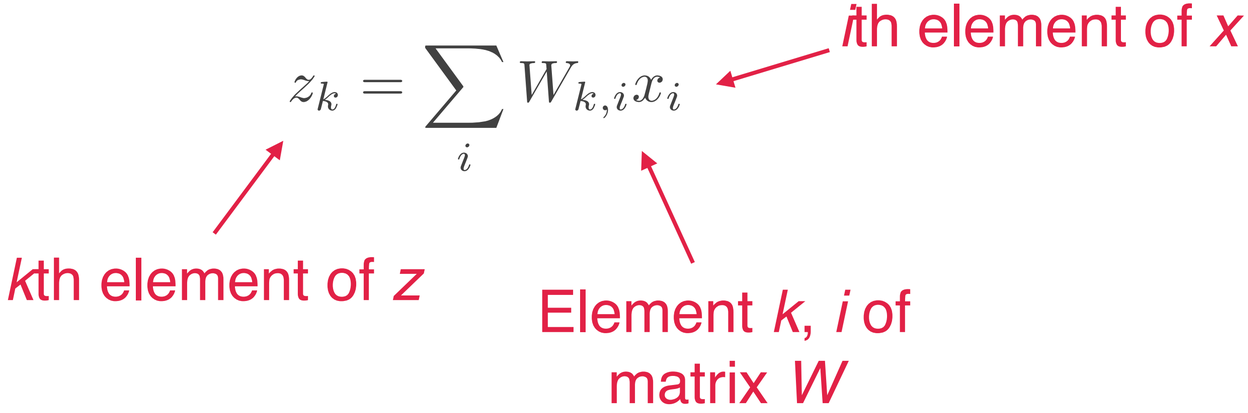


We “arrange” the weights as a matrix

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure:



Why convolutions?

- How can we change this?

  1. We maintain the spacial structure:

$$z_{k,\ell} = \sum_{i,j} W_{(k,\ell),(i,j)} x_{(i,j)}$$

i, j element of image *x*

(k, ℓ) element of *z*

TÉCNICO+
FORMAÇÃO AVANÇADA

Why convolutions?

- How can we change this?

    1. We maintain the spacial structure:

$$z_{k,\ell} = \sum_{a,b} W_{(k,\ell),(a,b)} x_{(k+a,\ell+b)}$$

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure

  2. To push for invariance, we make the weights independent of the input positions

$$z_{k,\ell} = \sum_{a,b} W_{(k,\ell),(a,b)} x_{(k+a,\ell+b)}$$

Depends on *k*, ℓ, *a* and *b*

TÉCNICO+
FORMAÇÃO AVANÇADA

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure

  2. To push for invariance, we make the weights independent of the input positions

$$z_{k,\ell} = \sum_{a,b} W_{(a,b)} x_{(k+a,\ell+b)}$$

Depends only on *a* and *b*

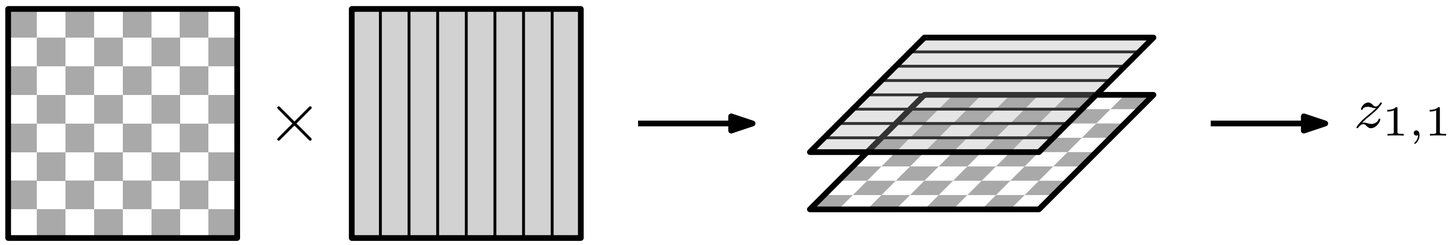
TÉCNICO+
FORMAÇÃO AVANÇADA

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure

  2. To push for invariance, we make the weights independent of the input positions

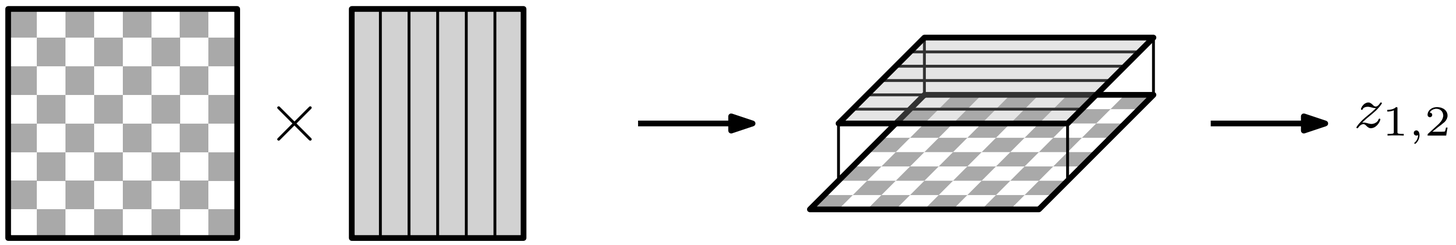


Why convolutions?

- How can we change this?

  1. We maintain the spacial structure

  2. To push for invariance, we make the weights independent of the input positions

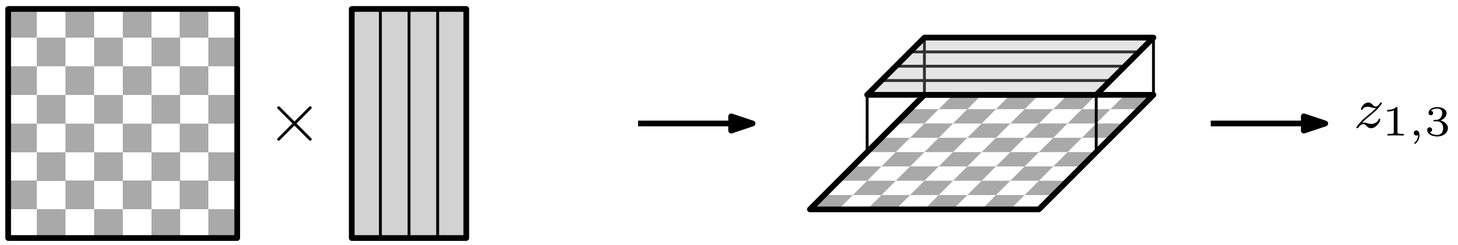


Why convolutions?

- How can we change this?

  1. We maintain the spacial structure

  2. To push for invariance, we make the weights independent of the input positions



Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

$$z_{k,\ell} = \sum_{a,b} W_{(a,b)} x_{(k+a,\ell+b)}$$

a and *b* have a small range

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

Why convolutions?

- How can we change this?

  1. We maintain the spacial structure
  2. To push for invariance, we make the weights independent of the input positions
  3. To push for locality, we make the weight matrices mostly 0

Why convolutions?

- The resulting operation is known as cross-correlation:

$$z_{i,j} = \sum_{a,b} W_{a,b} h_{(i+a,j+b)}$$

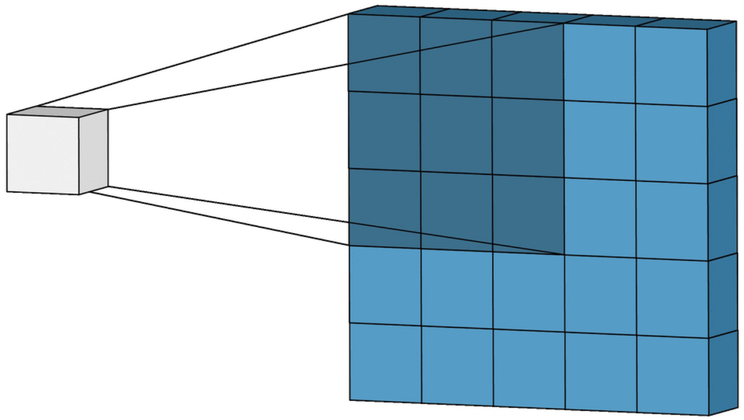
- If we flip the order of the weights, we can write, equivalently, the same operation as a convolution:

$$z_{i,j} = \sum_{a,b} W_{b,a} h_{(i-a,i-b)}$$

Since the weights are learned, the distinction is irrelevant, so we just refer to layers performing this operation as “convolutional layers”

Convolutional layers

- In a convolutional layer, a filter traverses the input and computes the output by performing point wise products



Convolutional layers

- In a convolutional layer, a filter traverses the input and computes the output by performing point wise products

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Filter (kernel)

1	0	1
0	2	0
1	0	1

Convolutional layers

- In a convolutional layer, a filter traverses the input and computes the output by performing point wise products

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Filter (kernel)

1	0	1
0	2	0
1	0	1

$$1 \times 1 + 3 \times 1 + 6 \times 2 + 9 \times 1 + 11 \times 1 = 36$$

Convolutional layers

- In a convolutional layer, a filter traverses the input and computes the output by performing point wise products

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Filter (kernel)

1	0	1
0	2	0
1	0	1

$$1 \times 1 + 3 \times 1 + 6 \times 2 + 9 \times 1 + 11 \times 1 = 36$$
$$2 \times 1 + 4 \times 1 + 7 \times 2 + 10 \times 1 + 12 \times 1 = 42$$

Convolutional layers

- In a convolutional layer, a filter traverses the input and computes the output by performing point wise products

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Filter (kernel)

1	0	1
0	2	0
1	0	1

$$1 \times 1 + 3 \times 1 + 6 \times 2 + 9 \times 1 + 11 \times 1 = 36$$
$$2 \times 1 + 4 \times 1 + 7 \times 2 + 10 \times 1 + 12 \times 1 = 42$$
$$5 \times 1 + 7 \times 1 + 10 \times 2 + 13 \times 1 + 15 \times 1 = 60$$

Convolutional layers

- In a convolutional layer, a filter traverses the input and computes the output by performing point wise products

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

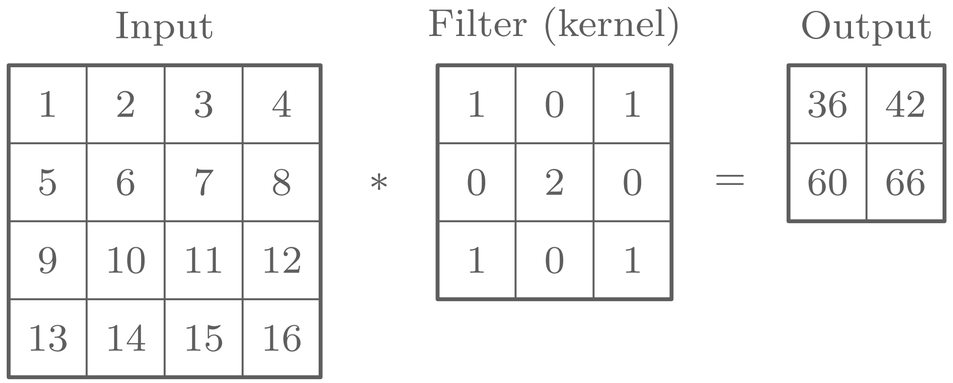
Filter (kernel)

1	0	1
0	2	0
1	0	1

$$1 \times 1 + 3 \times 1 + 6 \times 2 + 9 \times 1 + 11 \times 1 = 36$$
$$2 \times 1 + 4 \times 1 + 7 \times 2 + 10 \times 1 + 12 \times 1 = 42$$
$$5 \times 1 + 7 \times 1 + 10 \times 2 + 13 \times 1 + 15 \times 1 = 60$$
$$6 \times 1 + 8 \times 1 + 11 \times 2 + 14 \times 1 + 16 \times 1 = 66$$

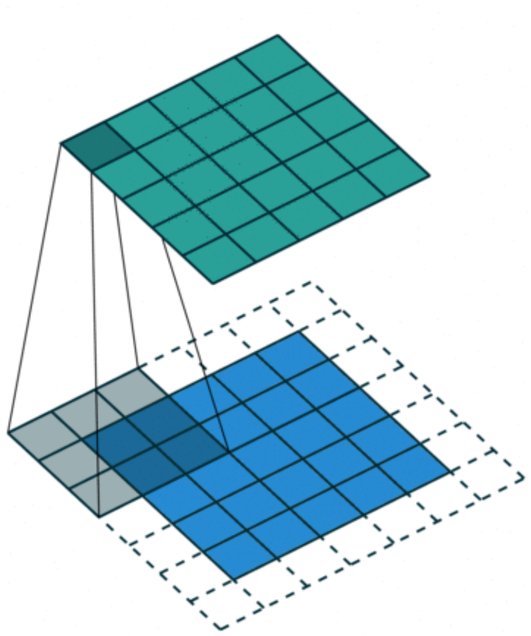
Convolutional layers

- In a convolutional layer, a filter traverses the input and computes the output by performing point wise products



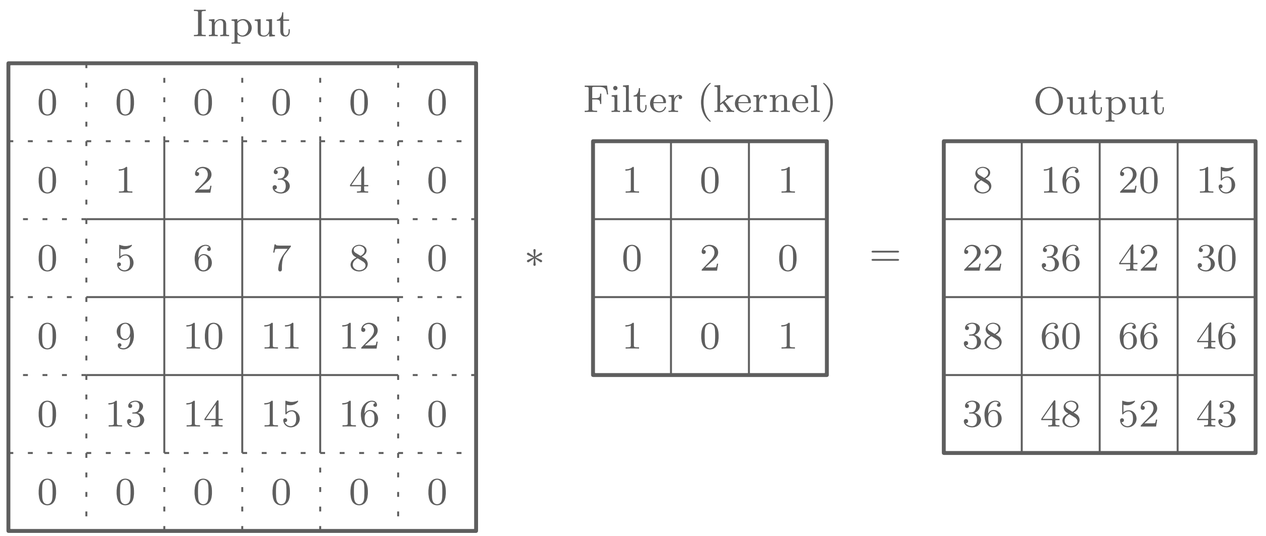
Padding and stride

- Padding refers to the addition of rows/columns of zeros to the image, in order to preserve (or not) the size of the resulting image



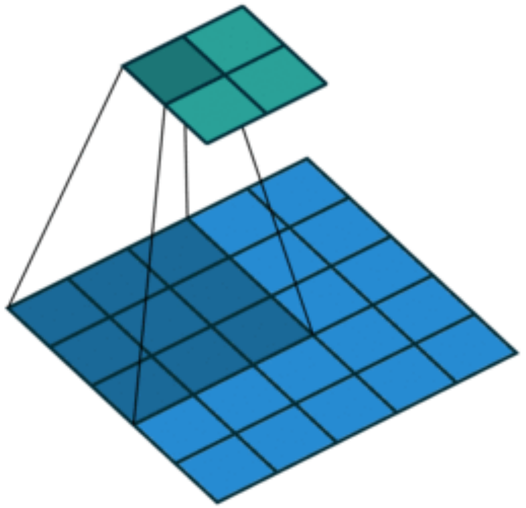
Padding and stride

- Padding refers to the addition of rows/columns of zeros to the image, in order to preserve (or not) the size of the resulting image



Padding and stride

- Stride refers to how many rows/columns are traversed in each “slide”
- In our examples so far, we used a stride of 1, but other values are possible

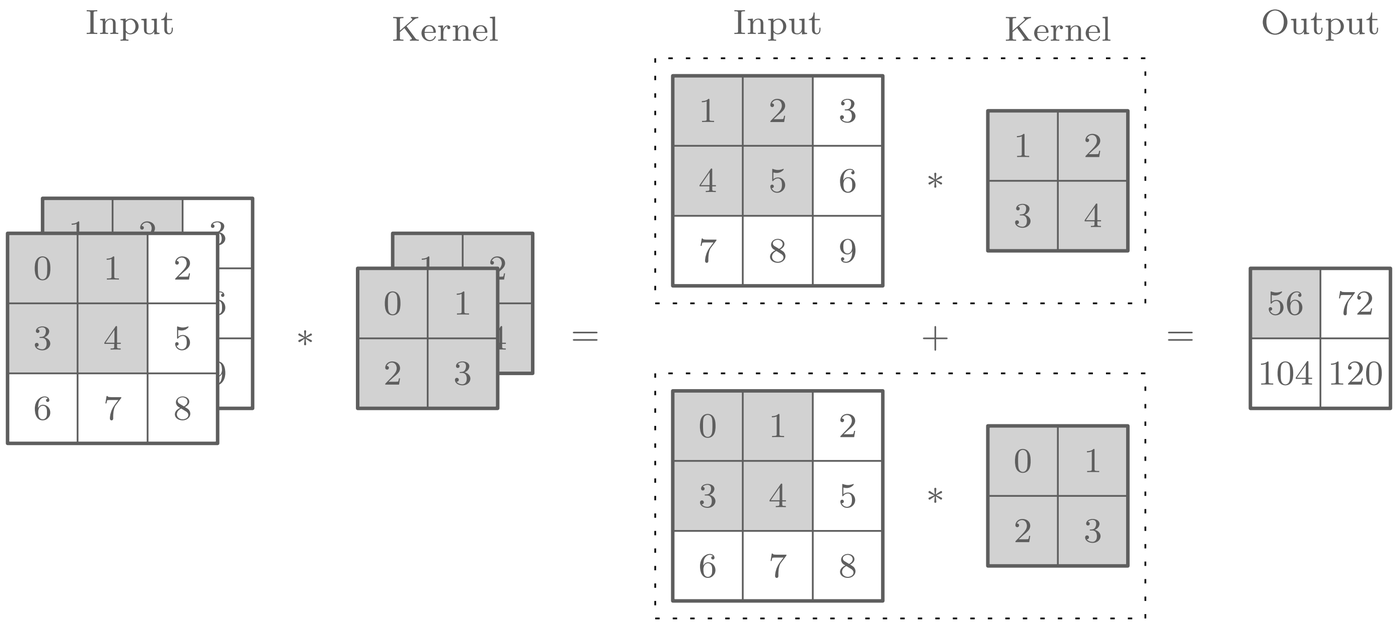


Channels

- So far, we considered that the input was a $L \times W$ image
- What if the image has multiple channels (e.g., RGB images)?
  - Convolution is applied to all channels
  - Each channel has its filter weights
  - The result is then added

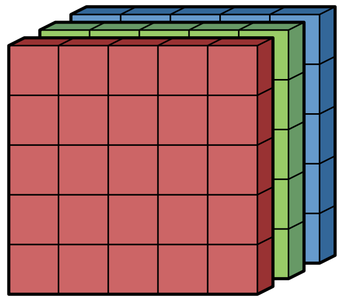
Channels

- So far, we considered that the input was a $L \times W$ image
- What if the image has multiple channels (e.g., RGB images)?

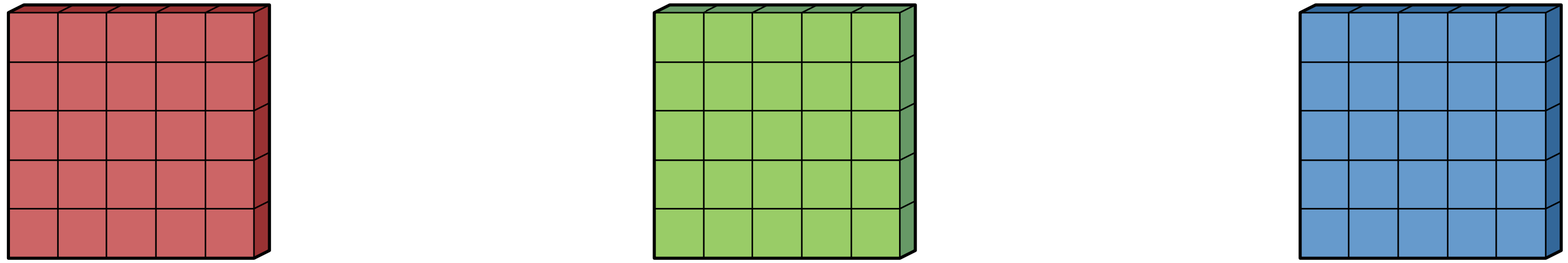


TÉCNICO+
FORMAÇÃO AVANÇADA

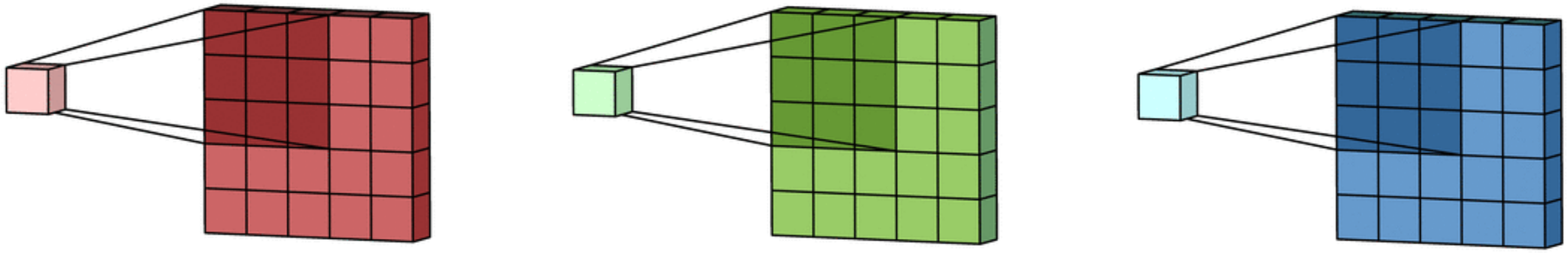
Channels



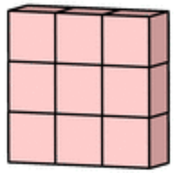
Channels

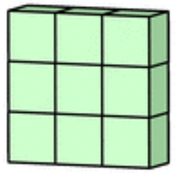


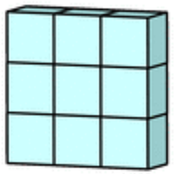
Channels



Channels







CNN “jargon”

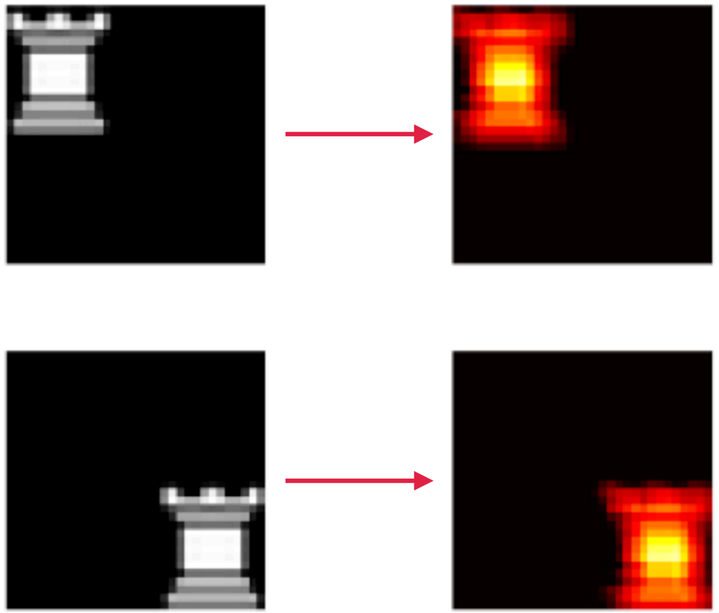
- Stride is the shift in rows/columns between two consecutive windows
- The number of channels is the number of filters considered in each layer
- Given an $N \times N \times D$ image, $F \times F \times D$ filters, K channels and stride S, the resulting output will be a $M \times M \times K$ image, with

$$M = \frac{N - F}{S} + 1$$

- Padding consists of “filling” the border with zeros (usually to maintain image size)

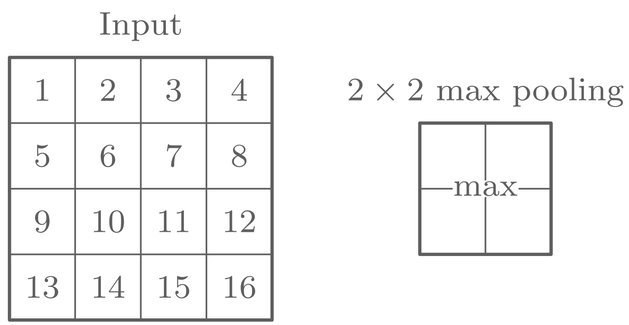
Equivariance

- Convolutional layers are equivariant (not invariant)
  - A shift in the input translates to a similar shift in the output



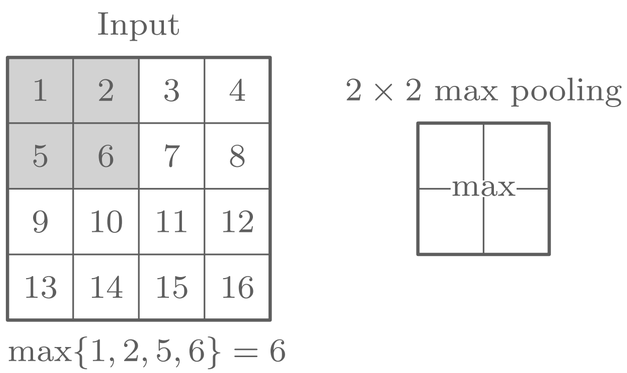
Pooling layers

- To further push for invariance, CNNs interleave pooling layers with convolutional layers
- A pooling layer is used to “summarize” the information from the previous layer



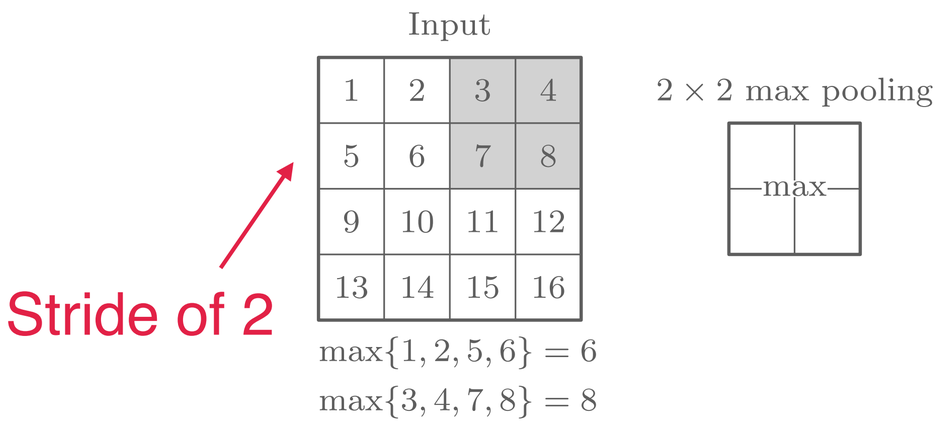
Pooling layers

- To further push for invariance, CNNs interleave pooling layers with convolutional layers
- A pooling layer is used to “summarize” the information from the previous layer



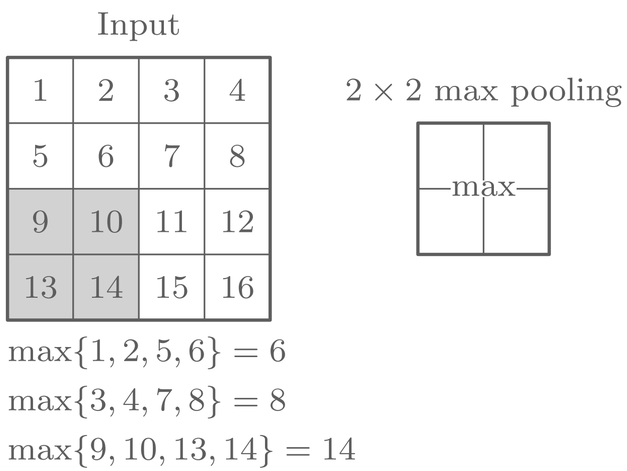
Pooling layers

- To further push for invariance, CNNs interleave pooling layers with convolutional layers
- A pooling layer is used to “summarize” the information from the previous layer



Pooling layers

- To further push for invariance, CNNs interleave pooling layers with convolutional layers
- A pooling layer is used to “summarize” the information from the previous layer



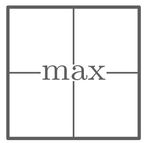
Pooling layers

- To further push for invariance, CNNs interleave pooling layers with convolutional layers
- A pooling layer is used to “summarize” the information from the previous layer

Input

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

2×2 max pooling



$\max\{1, 2, 5, 6\} = 6$

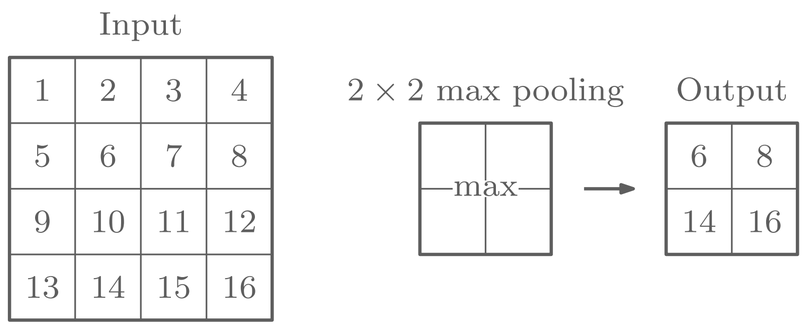
$\max\{3, 4, 7, 8\} = 8$

$\max\{9, 10, 13, 14\} = 14$

$\max\{11, 12, 15, 16\} = 16$

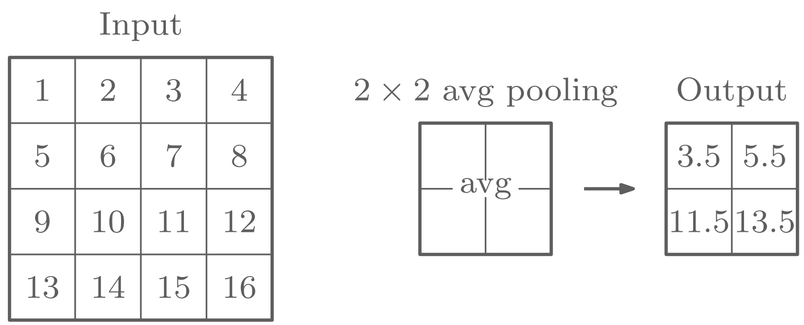
Pooling layers

- To further push for invariance, CNNs interleave pooling layers with convolutional layers
- A pooling layer is used to “summarize” the information from the previous layer



Pooling layers

- To further push for invariance, CNNs interleave pooling layers with convolutional layers
- A pooling layer is used to “summarize” the information from the previous layer

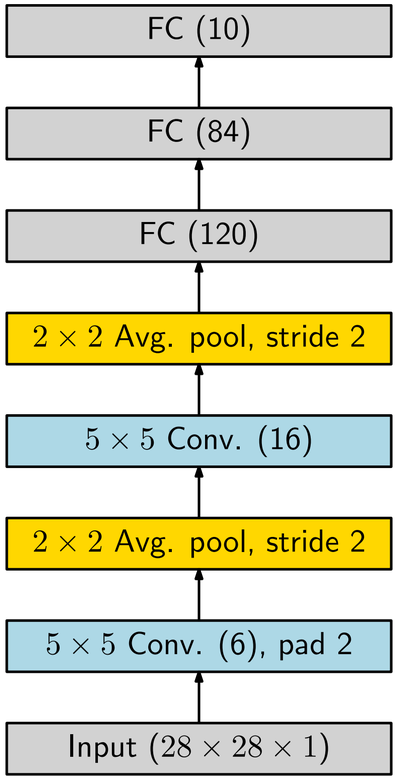


Pooling layers

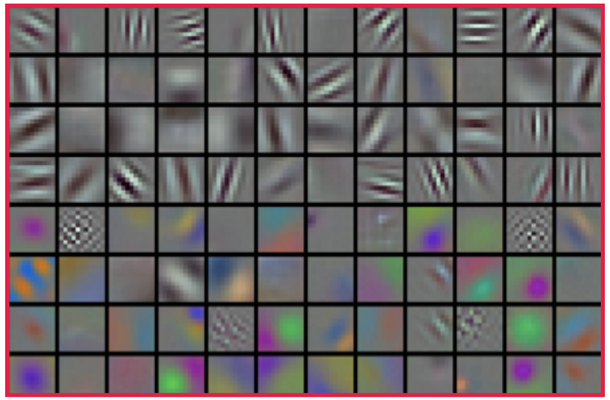
- To further push for invariance, CNNs interleave pooling layers with convolutional layers
- A pooling layer is used to “summarize” the information from the previous layer
  - Pooling layers exhibit local invariance
  - They are used to subsample images, to reduce computation

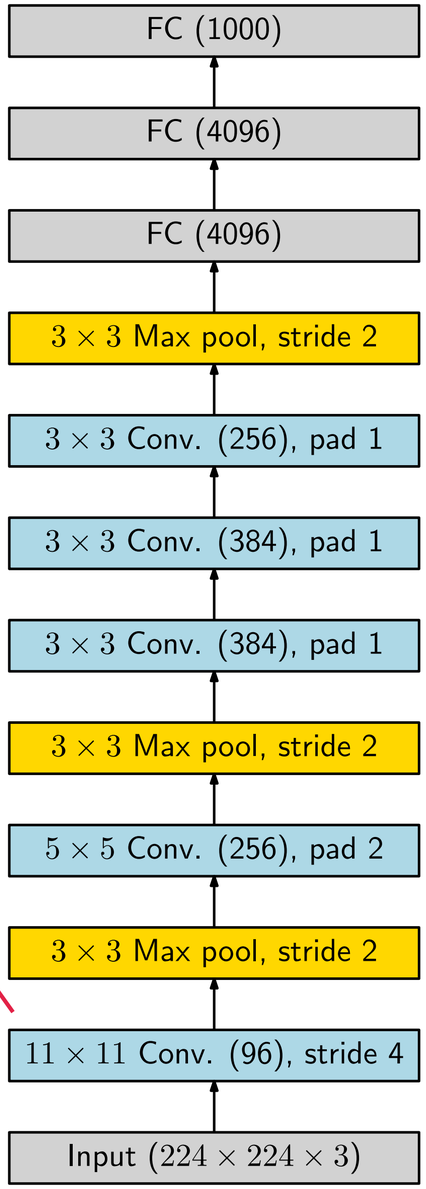
Let’s now go back a few slides…

First convolutional networks



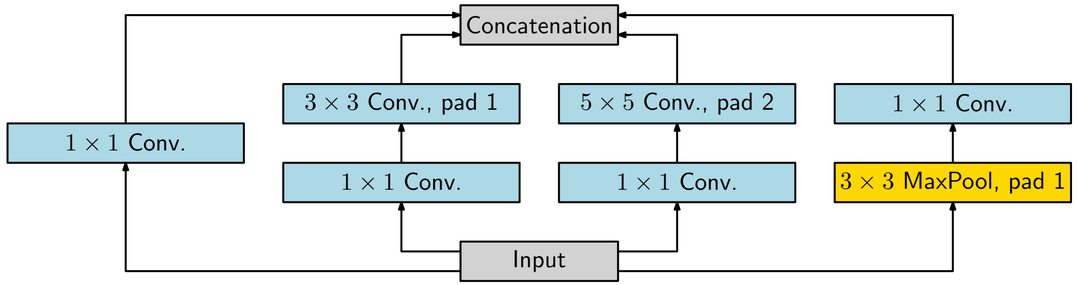
LeNet, 1998



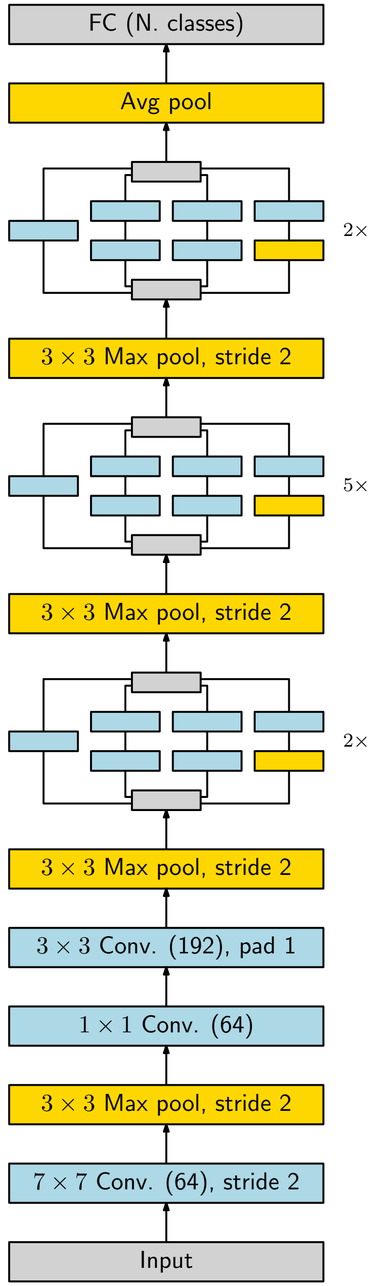


AlexNet, 2012

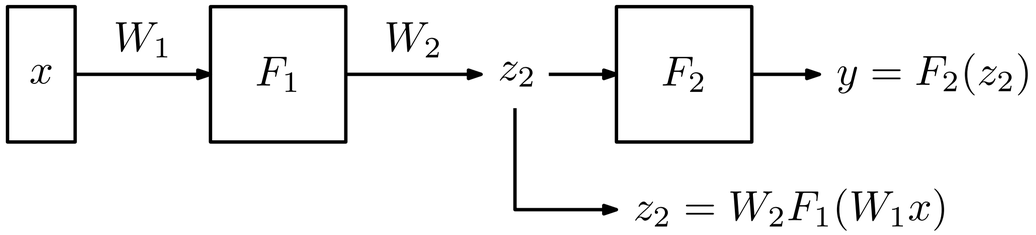
GoogLeNet (2015)



Inception block



Residual networks

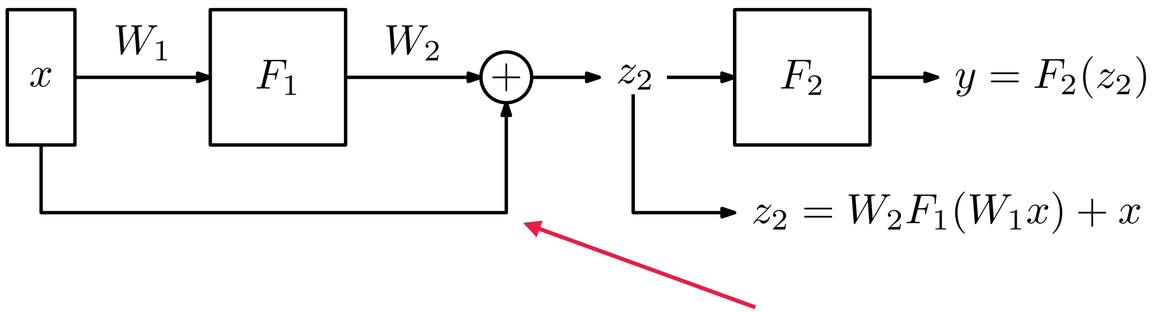


- Can we learn the function

$$y = F_2(x)?$$

- Maybe?

Residual networks

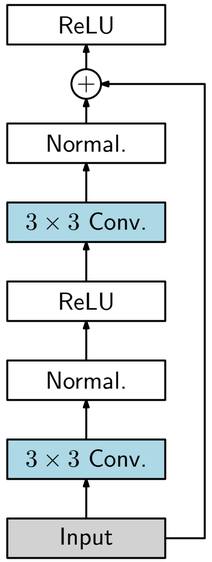


- Can we learn the function

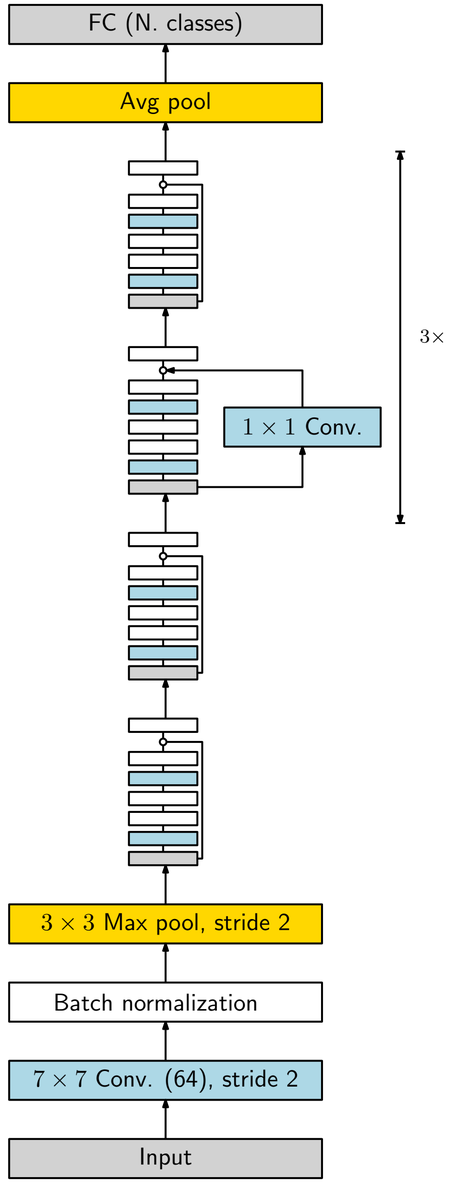
$$y = F_2(x)?$$

- Sure: We set $W_2 = 0$

ResNet-18 (2016)



Residual block



TÉCNICO+
FORMAÇÃO AVANÇADA

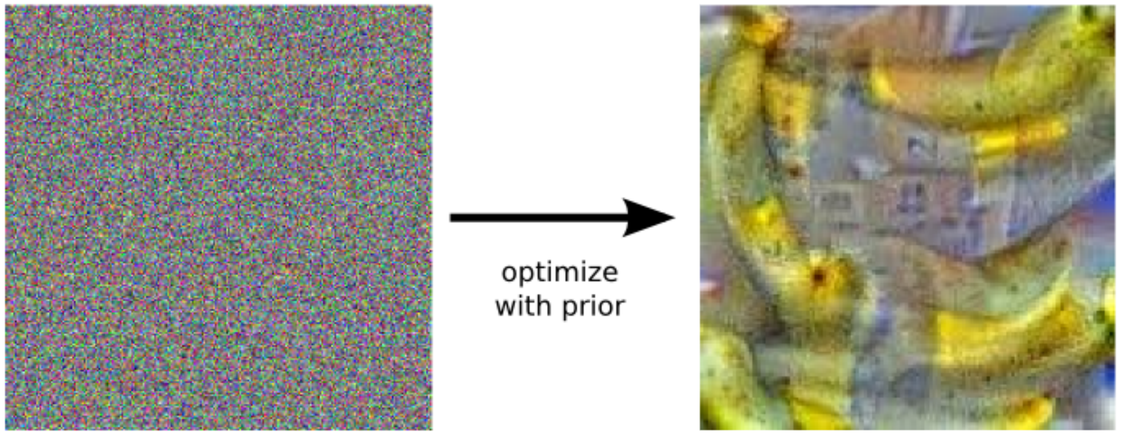
Final thoughts

Visualization

- The structure of convolutional layers allows for visualization of learned filters
- However, we can actually “invert” the network and try to “see” what’s happening in the network

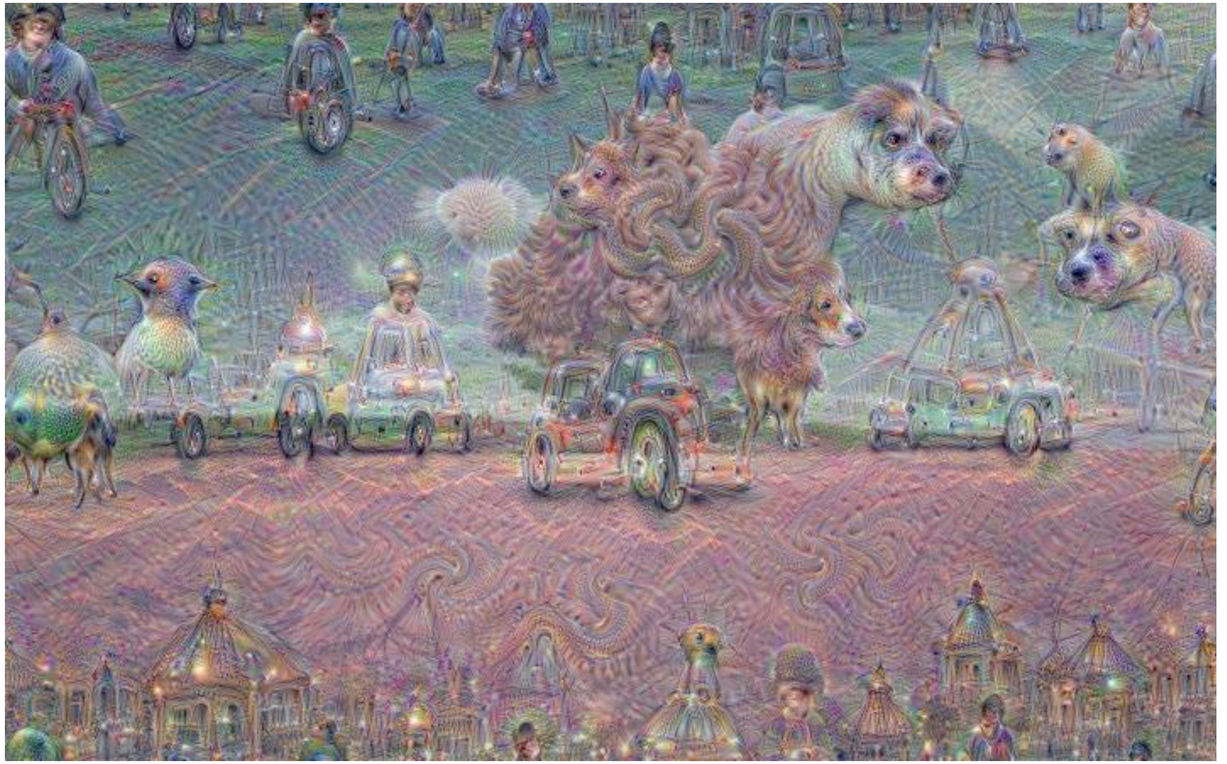
Visualization

- Idea: For a given class *c* (e.g., “banana”), optimize the input to maximize the probability of *c*
  - In other words, what is the input that yields the largest probability of being classified as a banana?



DeepDream

- We don’t have to give it noise—we can give it an initial image…



DeepDream

- We don’t have to give it noise—we can give it an initial image…

Adversarial attacks

- The same idea can be used to trick neural networks:
  - Given an input image, what's the smallest change to the image that fools the neural network?

One-pixel attacks

To conclude…

- **Key messages:**
  - Several algorithms exist to train neural networks
  - Different algorithms offer different advantages
  - Selecting the best algorithm for the task is often done by trial-and-error

To conclude…

Convolutional neural networks

- **Key messages:**
  - Convolutional neural networks are powerful architectures for processing structured inputs (e.g., images, sound)
  - CNNs typically comprise convolutional layers and pooling layers
  - Several interesting ideas (e.g., residual connections) have further pushed the power of CNNs

References

- Fukushima, K. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.” *Biological Cybernetics*, 36:193-202, 1980.
- Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.
- He, K., Zhang, X., Ren, S., and Sun, J. “Deep residual learning for image recognition.” In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 770-778, 2016.
- Hubel, D. and Wiesel, T. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex.” *J. Physiology*, 160:106-154, 1962.
- Kingma, D., and Ba, J. “Adam: A method for stochastic optimization.” *CoRR*, abs/1412.6980, 2014.
- Krizhevsky, A., Sutskever, I., and Hinton, G. “ImageNet classification with deep convolutional neural networks.” In *Adv. Neural Information Processing Systems*, pp. 1097-1105, 2012.

References

- LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., and others. “Gradient-based learning applied to document recognition.” *Proc. IEEE*, 86(11):2278-2324, 1998.
- Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. “Visualizing the loss landscape of neural nets”. In *Adv. Neural Information Processing Systems 31*, pp. 6391-6401, 2018.
- Polyak, B. “Some methods of speeding up the convergence of iteration methods.” *USSR Computational Mathematics and Mathematical Physics*, 4(5):1-17, 1964.
- Simonyan, K., Vedaldi, A., and Zisserman, A. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” In *Int. Conf. Learning Representations* (workshop poster), 2014.
- Su, J., Vargas, D., and Sakurai, K. “Attacking convolutional neural network using differential evolution” *IPSJ Trans. Computer Vision and Applications*, 11, article n. 1, 2019.

References

- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., and others. “Going deeper with convolutions.” In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 1-9, 2015.
- Tieleman, T., and Hinton, G. “Lecture 6. RMSProp: Divide the gradient by a running average of its recent magnitude.” In *Coursera: Neural Networks for Machine Learning*, 4(2):26-31, 2012.